Technical Report 1214

The Grasping Problem: Toward Task-Level Programming for an Articulated Hand

Nancy S. Pollard

MIT Artificial Intelligence Laboratory

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AI-TR 1214	2. GOVT ACCESSION NO
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): The Grasping Problem: Toward Task-Level Programming for an Articulated Hand	5. TYPE OF REPORT & PERIOD COVERED: technical report
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): Nancy S. Pollard	8. CONTRACT OR GRANT NUMBER(s): N00014-86-K-0685, N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: May 1990
	13. NUMBER OF PAGES: 93
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

robotics
robot hands
grasp planning
stability
feasibility
motion planning

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents a system for generating a stable, feasible, and reachable grasp of a polyhedral object. A set of contact points on the object is found that can result in a stable grasp; a feasible grasp is found in which the robot contacts the object at those contact points; and a path is constructed from the initial configuration of the robot to the stable, feasible final grasp configuration. The

(continued on back)

Block 20 continued:

algorithm described in the report is designed for the Salisbury hand mounted on a Puma 560 arm, but a similar approach could be used to develop grasping systems for other robots. Simulations show that the system can generate a wide range of grasps in difficult situations.

The Grasping Problem: Toward Task-Level Programming for an Articulated Hand

by

Nancy S. Pollard

This report was also submitted to the Department of Electrical Engineering and Computer Science of the Massachusetts Institute of Technology in May, 1989, in partial fulfillment of the requirements for the degree of Master of Science.

Acknowledgements: This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's research is provided in part by the University Research Initiatives under Office of Naval Research contract N00014-86-K-0685 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124. Personal support for the author was also provided by an Office of Naval Research Fellowship.

The Grasping Problem: Toward Task-Level Programming for an Articulated Hand

by

Nancy S. Pollard

Abstract

This report presents a system for generating a stable, feasible, and reachable grasp of a polyhedral object. A set of contact points on the object is found that can result in a stable grasp; a feasible grasp is found in which the robot contacts the object at those contact points; and a path is constructed from the initial configuration of the robot to the stable, feasible final grasp configuration. The algorithm described in the report is designed for the Salisbury hand mounted on a Puma 560 arm, but a similar approach could be used to develop grasping systems for other robots. Simulations show that the system can generate a wide range of grasps in difficult situations.

Thesis Supervisor: Tomás Lozano-Pérez
Title: Associate Professor, Department of Electrical Engineering and Computer Science

Acknowledgements

I would first of all like to thank my advisor, Tomás Lozano-Pérez, for his guidance, valuable discussions, and careful reading of drafts of this report.

I would also like to thank David Siegel, who helped me to clarify my ideas by making me explain them, and who also provided many helpful comments on drafts of the report.

Many others deserve mention: especially my friends, who made life interesting and were willing to look at my demos; ONR, who supported me with a fellowship during the last three years; and everyone else who helped to make the lab a challenging and exciting place to do research.

Contents

List of Figures

Chapter 1

Introduction

One goal of much research in robotics today is to develop the truly autonomous robot, a robot that can be used for undersea exploration, for exploration of planets, or for making repairs in remote or dangerous locations. Such a robot needs to have a lot of flexibility, a rich space of possible motions, since it must be able to perform not only complicated operations but also a *variety* of operations without special help.

In the context of manipulation, for example, it might be desirable to have a single, flexible gripper rather than carrying a specialized gripper for every operation to be performed. Some mechanisms that begin to satisfy this criterion have already been built. There are a number of robot hands available that are capable of performing a wide range of manipulation tasks (see [41], [14], [39]).

To use such a robot effectively, however, we need to be able to program it at a task level (at least at the level of, say, "get that rock" or "connect the two beams"), rather than providing it with a specific sequence of motions to perform. Task-level programming has been a research goal throughout the history of robotics and artificial intelligence, but we have yet to achieve this level of abstraction or autonomy.

An example of current state of the art in task-level programming is the Handey system [31], which can plan the task of picking up and relocating an object for an arm with a parallel jaw gripper. In this report, we will explore the problem of extending this system so that we can make use of a hand. Specifically, we will consider the three finger, three joint per finger Salisbury hand [41] (Figure 1.1). We will focus on the subtask of grasping, as the replacement of the parallel jaw gripper by the hand introduces many new potential ways to grasp an object. Our goal is to produce a relatively fast grasp planner that works in a variety of situations. We hope that detailed exploration of this specific problem will provide insight useful in solving more general grasping and manipulation

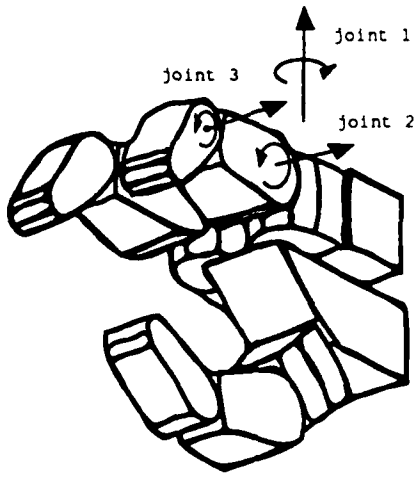


Figure 1.1: Model of the Salisbury hand.

problems.

1.1 The Problem

The problem addressed in this report is automation of the grasping task. We want to generate a path for a robot that will take it from a starting position to a stable grasp of a given object. To make the problem tractable, however, we consider a particular robot and make several assumptions about the world and about the types of grasps we want to achieve.

1.1.1 Problem Setting

Our robot consists of a Puma arm with a Salisbury hand mounted at the wrist (Figure 1.2). The arm has six degrees-of-freedom: the first three joints affect wrist position and wrist orientation; the other three change only the orientation. The hand has nine degrees-of-freedom: three joints for each of the three fingers.

Our assumptions about the world and restrictions on the task are as follows:

- Only fingertip grasps are considered.
- The fingertip contacts are modeled as hard finger contacts (that is, point contacts) with friction. This means that forces normal to and tangential to the object surface can be applied at the contact point (as long as the

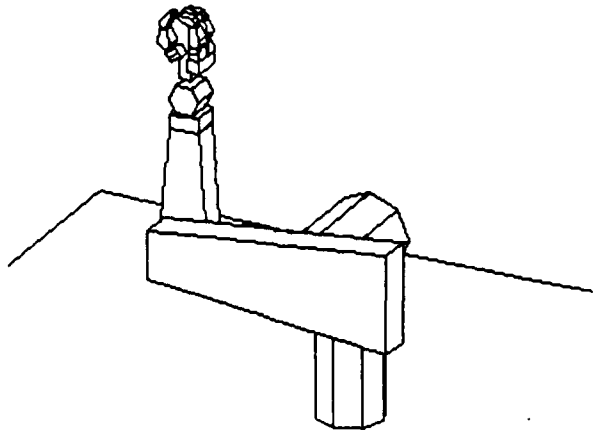


Figure 1.2: The Puma arm with the Salisbury hand.

resultant force is within the finger-object friction cone), but no torque can be applied about the contact point (Figure 1.3).

- There is a world model, and all objects in the world are modeled as polyhedra.

The restrictions on the grasp type, contact type, and object type give us a reduced problem domain to explore. The world model is used to detect collisions that would result from a proposed motion. The accuracy of our solutions depends on the accuracy of the world model.

With all this in mind, our goal becomes to generate a path for the Puma with the Salisbury hand that takes it from a starting position, through the modeled world, into a stable, fingertip grasp of a given object.

1.1.2 A Good Grasp

There are three requirements for a given grasp to be valid:

- Stability
- Feasibility
- Reachability

A grasp is stable if the forces on the object are in equilibrium and if small external *disturbance* forces do not cause the contacts to slip or separate from

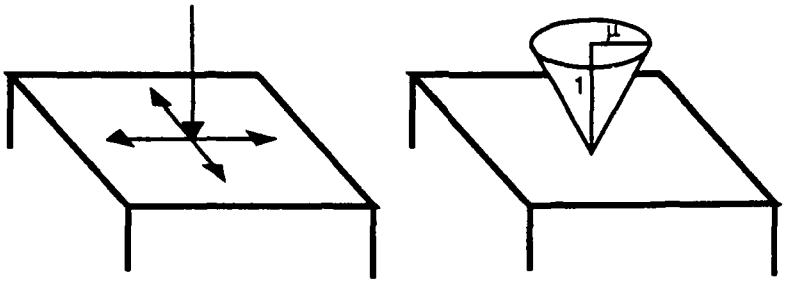


Figure 1.3: (left) With hard finger contacts with friction, we can apply forces normal to and tangential to the object surface at the point of contact. (right) But the resultant force must be within the friction cone at the contact point. We assume Coulomb friction, so the tangential force must be less than μ times the normal force, where μ is the coefficient of friction at the contact point.

the object. A grasp is feasible if there exists a collision-free configuration of the robot such that the fingertips are at the chosen contact points. The robot cannot be occupying the same space as some object in the world. A grasp is reachable if there exists a collision-free path from the starting position of the robot to the final grasp configuration. In fact, since we are interested in achieving the grasp, we not only want to know that such a path exists, but we want to find a path.

The overall plan of attack in this report is to first find a set of fingertip contact points (each with regions of allowable position error) that can result in a *stable* grasp; to then use the remaining degrees-of-freedom to generate a *feasible* grasp with the fingertips at those contact points; and then to establish *reachability* by identifying a good approach direction and synthesizing a path into the grasp. We will introduce each of these subproblems below. They will be discussed in more detail in Chapters 2 through 5.

1.1.3 Stability

Our goal for this subproblem is to find a set of contact points on a given object that can result in a stable fingertip grasp for a three-fingered hand. This is a six degree-of-freedom problem, as there are two degrees of freedom in placing each finger on the surface of the target object.

The first thing to note when considering this problem is that we do not have to worry explicitly about object *stability*. It is sufficient to achieve *equilibrium* of the contact forces on the object, provided that we have a sufficient number

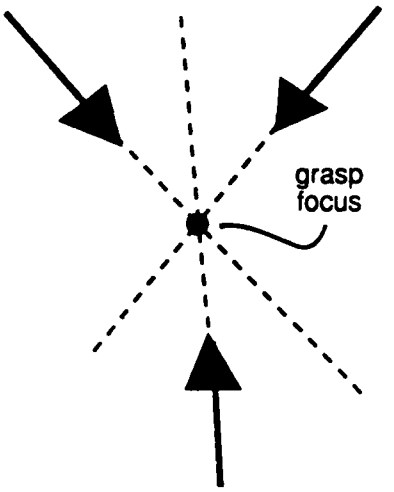


Figure 1.4: If we have three contact points, the contact forces must all meet at a point in the plane of the contact points for the grasp to be in equilibrium.

of contact points. Specifically, Nguyen [37] has shown for a polyhedron that any non-marginal equilibrium grasp, that is, any equilibrium grasp in which all contact forces are completely within the friction cones at the contact points (i.e. not on the friction cone edges) can be made stable by adjusting finger stiffnesses at the contact points if there are at least three hard finger contacts. We model the three fingers of our hand as hard finger contacts, so it is sufficient to synthesize a non-marginal equilibrium grasp for the three fingertips.

Another result drawn from the literature is that, if there are forces applied at three contact points, a necessary condition for equilibrium of these forces is that their lines of direction all meet at a point in the plane of the contact points [3] (see Figure 1.4). This point can be at infinity, which means that the lines of force direction are all parallel. We will call the point of intersection the *grasp focus* (as in [3]).

Since all of the forces applied at the contact points must lie within the plane of the contact points, it is natural to break the problem into finding this plane and then placing the contacts within the plane. This is the approach we will take.

Considerations involved in placing both the plane and the contact points are to keep the forces within the friction cones at the contact points and to allow for position error in placement of the fingertips. The need to allow for position error

might keep us from placing the contact points on vertices or edges, for example, although they might be otherwise desirable because of their large friction cones. We will place the contact points on the object faces, and attempt to pursue Nguyen's [37] goal of producing maximal independent contact regions for the fingertips.

Note that the considerations used to determine a good set of contact points represent an effort to avoid any chance of the object slipping in the grasp, even if there is some position error in the final placement of the fingertips. It is worth pointing out that slipping may not be a problem to be avoided at all costs (see for example Fearing [17] and Brock [3]). We can do a quasistatic analysis of an initial contact situation that takes slipping into account, and with this, we can estimate the final positions of the object and the fingertips given a method of control, the assumption of very slow motion, a good object model, and a predictable support surface (see Mason [34]). Here we assume that slipping is not substantial and that such an analysis is not necessary.

Also note that when we find contact forces and a grasp focus, we ignore known external forces such as gravity. Since we will have a stable grasp of the object, gravity and other disturbance forces can be counteracted without the object slipping from the grasp, although the contact forces may have to be large if the object is heavy or if the coefficient of friction is small.

1.1.4 Feasibility

A second subproblem is to find a collision-free configuration of the robot with the fingertips at the selected contact points. Since we have already specified three fingertip contact positions (nine degrees-of-freedom), we have six remaining degrees-of-freedom. We can specify these by picking a wrist position and orientation.

Our method is to first ignore the objects in the world and choose a wrist configuration that is *kinematically* feasible and approximately centered about the fingers. If the resulting arm/hand configuration does result in collisions between the robot and objects in the world, we set up a quasistatic spring model of the joints of the robot and assume that there are forces at the points of collision that nudge the robot away from the collision points (and hopefully, into free space).

We use this physical model of the joints and of collision forces in an attempt to make the resulting behavior somewhat predictable. For other options see [25], [24], [36].

1.1.5 Reachability

Our goal here is to find a path for the robot from its starting configuration to a goal configuration. This is a fifteen degree-of-freedom problem, as we need to find a path for all of the joints of the robot. To cut down on the number of degrees-of-freedom we need to consider at any one time, we break the problem into two parts:

- Near reachability.
- Global reachability.

For near reachability, we initially consider potential collisions between the hand and the target object or nearby objects. That way, we can plan motions of the hand alone. Furthermore, we assume that the hand is sufficiently close to the object and that the environment is sufficiently uncluttered that the hand can make a straight line approach to the target object. Unforseen collisions between the hand or the arm and objects in the world will be eliminated using the joint spring control method outlined in the *Feasibility* section above.

For global reachability we only worry about collisions that might be caused by making large arm motions. For this, we consider only motions of the first three joints of the arm, those responsible for the *position* of the wrist. The hand can be modeled as a fixed or infrequently varying payload. This will only work if there is a wide path for the robot to follow.

1.2 Previous Work

In this section, we give an overview of previous work related to the subproblems of finding a stable, feasible, and reachable grasp. Much of the work is specific to either two-fingered hands (e.g. parallel jaw grippers) or hands with three or more fingers.

In the area of stability, much work has been done in the analysis of stable grasps, both for two-fingered hands (see Barber et al. [2]) and for hands with three or more fingers (see for example Mishra et al. [35], Jameson [21], Cutkosky [11], Kerr and Roth [22], Salisbury [41], and Salisbury and Craig [42]).

We are more interested in grasp synthesis, however. Work in this area for hands with three or more fingers includes that of Hanafusa and Asada [19], who find a grasp on a two-dimensional object cross-section by minimizing the energy stored in springs at the contact points; Baker, Fortune, and Grosse [1] who show that a stable grasp of a polygon can be formed by placing contacts at points of intersection of the polygon with the maximum radius circle that can

fit inside it; Markenscoff and Papadimitrious [33], who optimize a stable grasp of a polygon with respect to the compressive forces required to balance external forces; Jameson [21], who maximizes a goodness function for grasp stability once the robot is in contact with an object; and Nguyen [37], who discusses finding maximal independent contact regions that can produce a stable grasp of an object.

Another approach to choosing a grasp is to generate a catalogue of grasp types and to then match these grasp types to the target object or to the task to be achieved. Examples of this are Tomovic et al. [45], who have a system for matching grasp types to object types, and Li and Sastry [27], who match grasp types to task ellipsoids.

In the area of feasibility, most of the work has concentrated on finding free contact and approach areas around an object for a two-fingered hand. Examples of this include Pertin-Troccaz [40], Laugier [26], Wingham [46], and Lozano-Pérez [29, 28].

In the area of reachability, work has been done for two-fingered hands near the object, using a potential-field type approach (see Lozano-Pérez et al. [31]) and a configuration-space approach (see Pertin-Troccaz [40]).

Work on reachability that is not specific to hands includes the entire field of path planning. The standard approach for a global path planner is to first characterize free space for the robot, and to then perform a search through the characterization of this space. One might also map free space as the search proceeds. A goal is to do a fast, yet accurate characterization or mapping of free space that leads to an easy search (at least for simple problems). Much work in this area has been devoted to finding some way to characterize free space in a manner such that the difficulty of the resulting search accurately reflects the complexity of the task of moving the robot through the given environment. Examples of exact characterizations of free space include the Voronoi diagram [9, 38], Canny's roadmap algorithm [8], Schwartz and Sharir [43]. Examples of heuristic path planners include Lozano-Pérez [30], who divides configuration space into slices, Brooks [5, 4], who finds freeways in free space using generalized cones, and Faverjon [16], who uses an octree representation of free space.

Work on local path planning is also relevent. In this area, Khatib [23] combines an attractive potential toward the goal with repulsive potentials from nearby objects to move a robot. Faverjon and Tournassoud [15] combine an attractive force toward the goal with *constraints* imposed by the obstacles. Klein [25], Kirćanski and Vukobratović [24], and Nakamura et al. [36] begin with an underspecified path, which might be, for example, a set of consecutive wrist or fingertip positions for an arm. They then use some representation of the

goodness of configurations to find at each point along the path the best configuration that corresponds to the given specifications or path requirements. For all of these methods there is no guarantee that a path will be found.

Concerns that are important, but that will not be dealt with explicitly in this report are contact dynamics and uncertainty. The first comes up because we do not start in a stable grasp position, and so we need to think about what happens when we make contact with the object. In this vein, some work on analyzing object motion due to pushing and slipping in the grasp has been done by Mason [34], Brost [6], and Fearing [17].

The second concern is with producing a guarantee that a plan we generate will work in the presence of uncertainty in knowledge of the world and uncertainty in our ability to follow a given trajectory exactly. Work on guaranteed motion strategies can be found in Donald [12], Buckley [7], Erdmann [13], and Lozano-Pérez et al. [32], who develop back projections [7, 13, 32] and error detection and recovery schemes [12].

Solutions to many of the subproblems that are addressed here have been pulled together and implemented on a Puma arm with parallel jaw grippers [31]. The primary goal of this report is to address the additional concerns involved when the robot has a three-fingered hand. Thus, the new problems with respect to [31] are how to select a grasp and how to approach that grasp when we have a manipulator with a large number of degrees of freedom.

1.3 Contributions of this Report

In this report:

- We present an algorithm for synthesizing a stable, feasible, and reachable grasp of an object with an arm and articulated hand.
- In the area of stability we follow up on Nguyen's goal of generating maximal independent regions for fingertip contact on the object.
- In the area of feasibility we present an algorithm that first suggests a kinematically feasible configuration and then modifies it to avoid collisions.
- In the area of a grasp approach we modify a straight line approach to a final grasp using a potential field model of collision forces. Here and in the feasibility algorithm we use a quasistatic spring model for the joints of the robot to produce a somewhat predictable motion away from collisions.

- In the area of path planning, we present a parallel global path planner that works in the space of the first three joints of our robot. The path planner uses a precomputed map from configurations to Cartesian space regions to compute free configuration space.

1.4 The Rest of the Report

In the chapters that follow, we will first discuss the subproblems involved in synthesizing a valid grasp. Chapter 2 covers stability, Chapter 3 feasibility, Chapter 4 near reachability, and Chapter 5 global reachability. In Chapter 6, we present a summary of the process and discuss some possible extensions. In Chapter 7 we present conclusions.

Chapter 2

Stability: The Contact Points

In this chapter, we outline an algorithm for generating a stable grasp of a given object. We want to define three contact regions on the object from which we can generate a stable, fingertip grasp for our three-fingered hand. Our goal is to maximize the size of the smallest of these contact regions so as to minimize our chances for failure due to error in placement of the fingertips. We assume that we have only polyhedral objects in our world, and that our contact regions will be on faces of these objects.

The development of our algorithm is based on the fact that the hand we are using is designed with a finger configuration that consists of the two upper fingers opposing the thumb. Because of this, we focus on configurations of faces of our target object in which two faces roughly oppose a third. We seperate these configurations into four classes, which are treated somewhat differently:

- Three parallel faces. (This includes the case where we have two parallel faces, one having two contact points.)
- Three *nearly* parallel faces. (This includes all other configurations with only two faces.)
- Three faces that can make up a convex object.
- Three faces that must form a concave object.

Examples of each of these configurations are shown in Figures 2.1 through 2.4. The contact faces are highlighted, and a grasp is shown for each object, with the fingertips contacting the highlighted faces. The two upper fingers of the hand do not necessarily have to contact the two "upper" faces as in these examples, although this configuration will often be the best fit to the kinematics

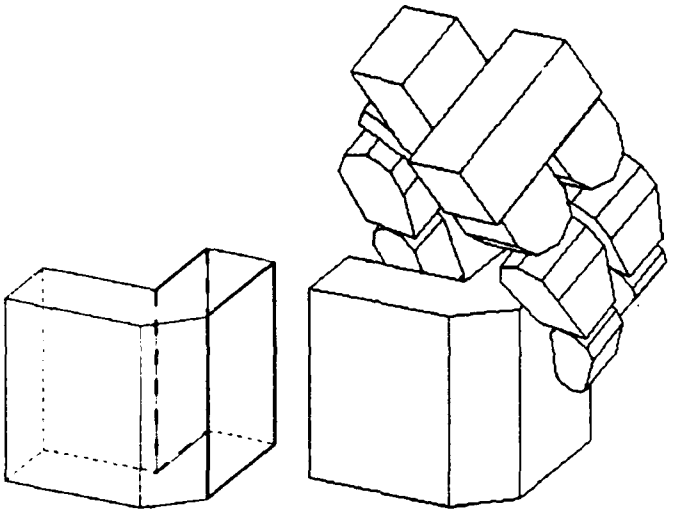


Figure 2.1: Two parallel faces.

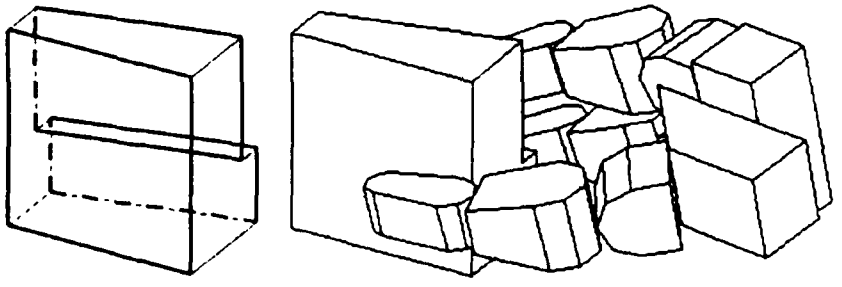


Figure 2.2: Three nearly parallel faces.

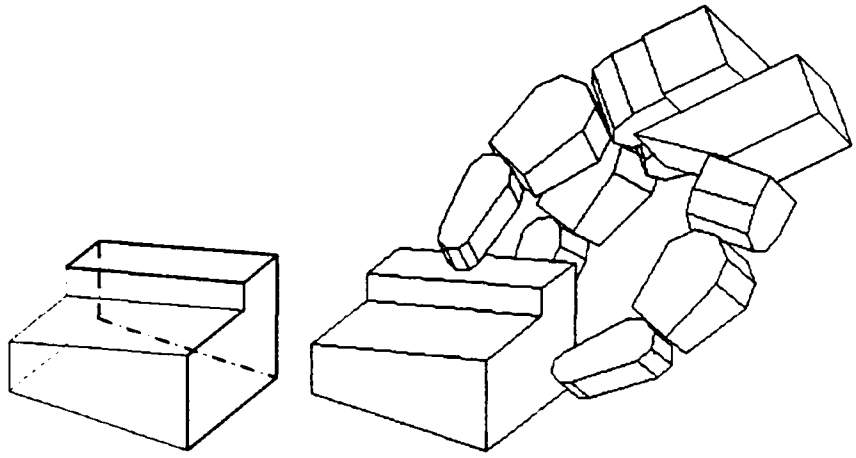


Figure 2.3: Three faces that can form a convex object.

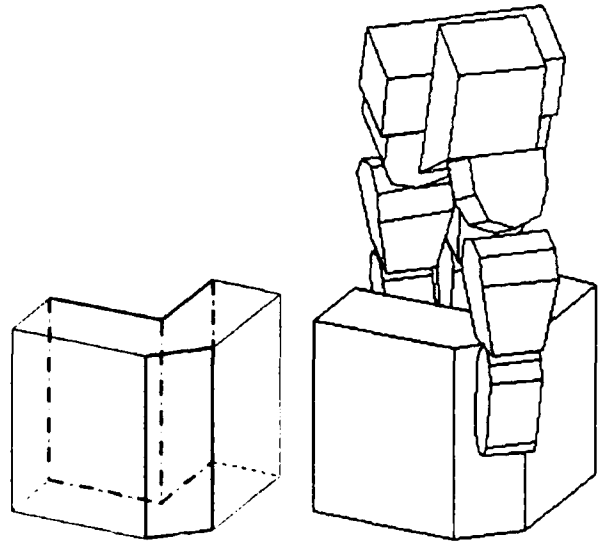


Figure 2.4: Three faces that form part of a concave object.

of the hand. The *Examples* section below illustrates some grasps with different finger configurations, such as the left finger and thumb opposing the right finger.

The following definitions are used in describing the algorithm for choosing contact points:

- **Finger-to-face mapping:** a mapping that indicates the face that each finger contacts.
- **Grasp plane:** the plane defined by the contact points.
- **Focus point:** the point in the grasp plane at which all the lines of contact force intersect.
- **Contact points:** the points at which the fingers contact the target object.
- **Contact regions:** the regions about the contact points such that if each finger contacts the object within its contact region, we can generate a stable grasp of the object.

In this chapter, we assume that a finger-to-face mapping is given (although see the *Stability Extensions* section of the *Discussion* chapter for ideas on how it could be selected). Our goal throughout is to maximize the size of the smallest valid contact region surrounding a contact point. In the sections below we discuss the algorithm for generating a stable grasp in detail, using the objects illustrated in Figures 2.1 through 2.4 as representative examples.

2.1 Grasp Plane

We assume that we are given a finger-to-face mapping, and our goal in this chapter is to find contact points for the fingers on each of their respective faces. This is a six-dimensional problem, since for each of the three fingers we have a two-dimensional surface on which we can place the contact point for that finger. We know, however, that because we have only three contact points, the forces applied at the contacts must all meet at a point (the focus point) if they are to be in equilibrium [3]. This point must lie in the plane of the contacts, so we can easily break our problem down into selecting a grasp plane and then selecting contact points within that plane. There are three degrees-of-freedom in choosing a grasp plane, which can be specified as an orientation and offset. This leaves three remaining degrees-of-freedom with which to place the contact points on the edges formed from the cross-section of the contact faces in the grasp plane.

In this section, we discuss choosing an orientation and offset for a grasp plane. In choosing the plane orientation, we would like to maximize the minimum size friction-cone, or effective coefficient of friction, *in* the grasp plane for the three faces. Having the largest friction-cones possible helps us to achieve the goal of maximizing the size of the smallest valid contact region for the fingertips. In choosing the grasp plane offset, we use only a simple heuristic suggested by the hand kinematics.

2.1.1 Orientation

We know that the fingertip contact forces will lie within the grasp plane we choose. It is a useful concept, therefore, to define an *effective coefficient of friction* for the cross-section of each contact face within the grasp plane (that is, for each *contact edge*). This captures the idea that only a slice of the friction cone of the contact face is seen in the grasp plane. It will allow us to reduce the problem of finding contact points to a planar problem. The contact faces are reduced to contact edges, each with an effective coefficient of friction. Note that for polyhedra the effective coefficient of friction depends only on the orientation of the grasp plane, not on its offset, or location in space.

We assume that the real coefficient of friction is the same for all faces and is equal to μ. If the grasp plane is at angle θ_i from the face normal, then it slices the friction cone so that the effective coefficient of friction is (see Figure 2.5):

$$\frac{\sqrt{\mu^2 - \tan^2 \theta_i}}{\sec \theta} \tag{2.1}$$

The effective coefficient of friction of a contact edge is largest (and equal to μ) when the normal of the corresponding face lies in the grasp plane. In choosing a grasp plane orientation, we wish to maximize the size of the smallest effective coefficient of friction in the grasp plane. We can say that this defines a *natural* plane orientation for any configuration of face normals.

Natural Orientation: To characterize the natural plane orientation for a given configuration of faces, we look at the rank of the matrix N composed of the normal vectors of the three faces:

$$N = \begin{bmatrix} n_{1x} & n_{1y} & n_{1z} \\ n_{2x} & n_{2y} & n_{2z} \\ n_{3x} & n_{3y} & n_{3z} \end{bmatrix} \tag{2.2}$$

- If $rank[N] = 1$, then all three faces are parallel, and we have a one degree-of-freedom range of grasp plane orientations possible in which the grasp

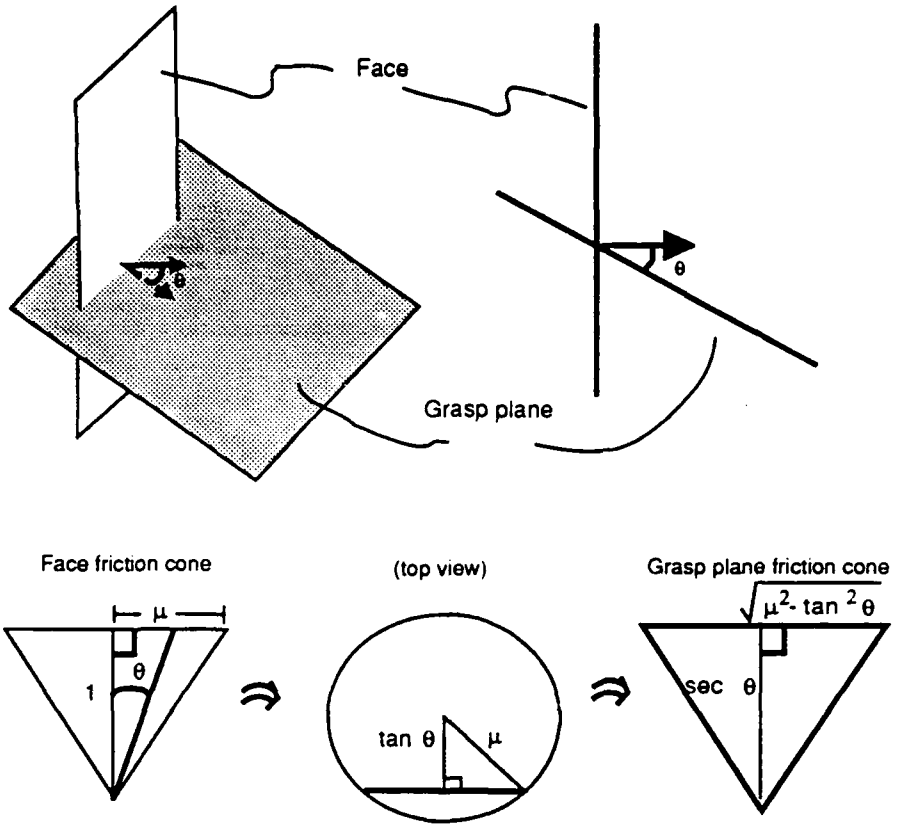


Figure 2.5: The grasp plane cuts the face at an angle of θ from the face normal. The corresponding cut in the friction cone of the face reduces the effective coefficient of friction in the plane to $\sqrt{\mu^2 \cos^2\theta - \sin^2\theta}$.

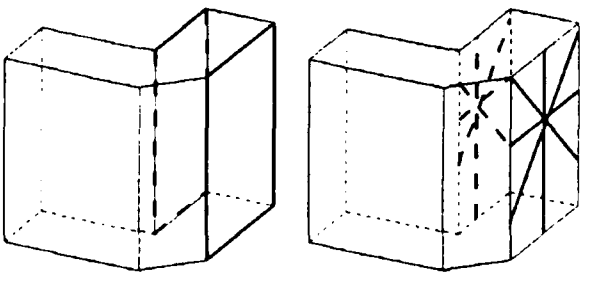


Figure 2.6: All the faces are parallel, so we have a range of possible grasp planes.

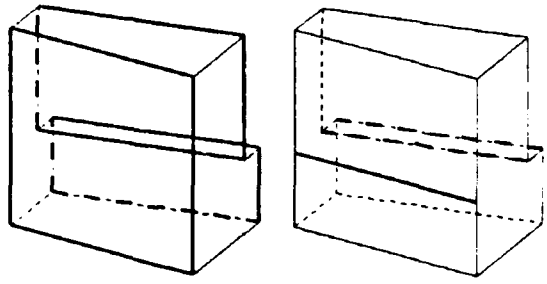


Figure 2.7: The two upper faces are parallel, so we have only two independent normals. The grasp plane shown is formed using those normals as basis vectors, and it is the unique plane perpendicular to all three faces.

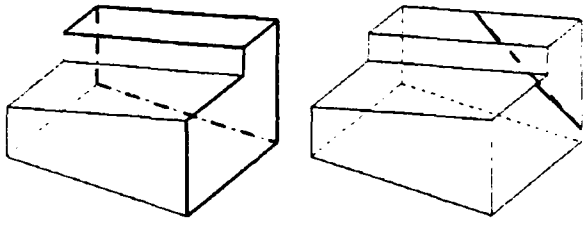


Figure 2.8: The three faces have linearly independent normals, so we fit a plane to them that gives us equal angles of deviation of the normals from their projections onto the grasp plane.

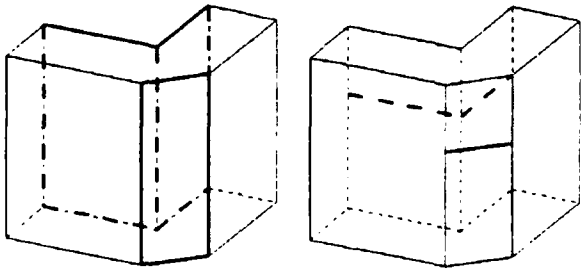


Figure 2.9: This is another grasp plane formed using two independent normals as basis vectors.

plane is perpendicular to all three faces and so all three effective coefficients of friction are at their maximum possible values (μ). Figure 2.6 shows some of the grasp plane orientations possible for one such configuration.

- If $rank[N] = 2$, then only two of the faces have independent normals, and we can choose a unique grasp plane orientation (using these normals as basis vectors for the plane) in which the plane is perpendicular to all three faces, and the effective coefficients of friction are all, again, μ. This case is shown in Figures 2.7 and 2.9.

- If $rank[N] = 3$, then we have three faces with independent normals, so we cannot find a plane perpendicular to all three faces. We settle here for finding the unique plane orientation in which the three effective coefficients of friction will be equal. To do this we define our plane as $ax + by + cz = 1$, and solve the following matrix equation for $[abc]^T$:

$$N \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{2.3}$$

  Vector **a** ($[abc]^T$) is the normal of the grasp plane (generally not a *unit* normal), and n_i (row i of N) is the unit normal of face i, so equating ($n_i \cdot a$) to 1 for all faces ensures that the angles of face normals from the grasp plane will all be equal, and hence that the effective coefficients of friction in the plane will all be equal. Because all of our contact points cannot be in the same half of the circle formed from the contact points (we could not grasp anything without some opposing force), changing the orientation of the grasp plane in any direction would cause one of the angles of face normals from the grasp plane to become larger, and thus cause one of the effective coefficients of friction in the plane to become smaller. This implies that with our calculated grasp plane, we have achieved our goal of maximizing the minimum size effective coefficient of friction. Figure 2.8 shows an example grasp plane for this type of configuration.

Leeway: The above constraints define a natural orientation for the grasp plane. We can also define another concept, the *leeway* of a particular orientation. Intuitively, this is the amount that the orientation of the grasp plane can *change* before the contact forces can no longer point into the plane without the grasp slipping, that is, before one of the effective coefficients of friction would go to zero. Leeway can be defined as the ranges of valid rotations of the original grasp plane about axes within that grasp plane. The orientations of the axes, and thus the leeway, depend on a single parameter. In other words:

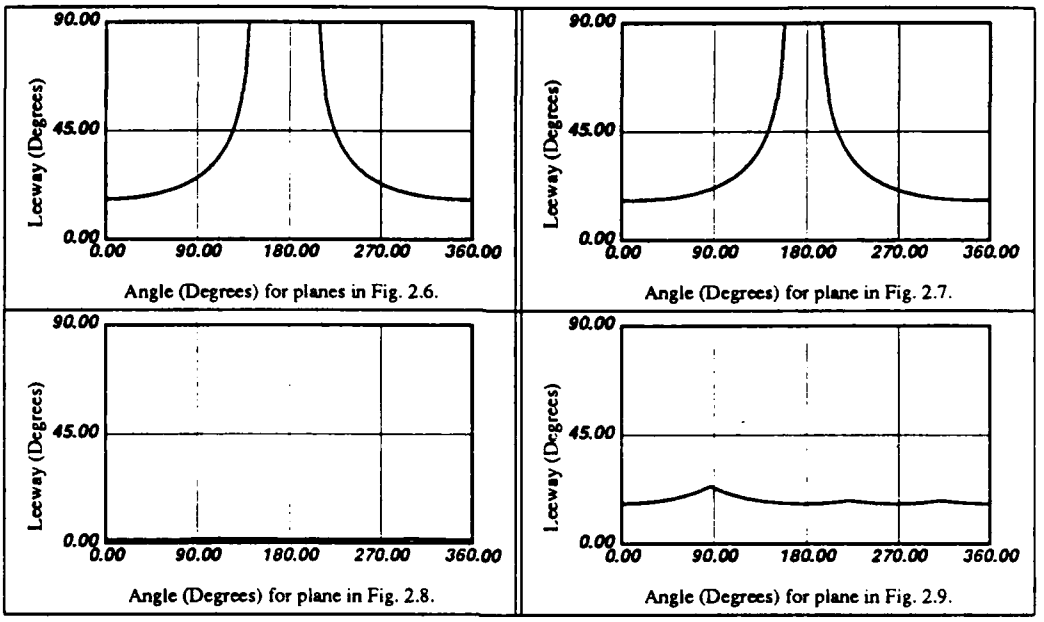


Figure 2.10: Plots of the leeway in orientation of the grasp plane as a function of the line about which that plane is rotated.

- Define an axis of rotation in the grasp plane. Since we are only concerned about the *orientation* of the grasp plane, only the direction of this axis matters, so it is a function of one parameter, say angle α.

- Leeway(α) is the minimum angle that the grasp plane can rotate about this axis before the effective coefficient of friction on some face goes to zero.

Plots of leeway versus angle α are shown in Figure 2.10. The plots correspond to the leeway in the grasp planes shown in Figures 2.6 through 2.9. Note that if the leeway about a particular axis reaches 90 degrees (that is, the plane can be rotated at least 90 degrees in either direction about the chosen axis), a valid grasp plane can be placed at *any* orientation about that axis.

To understand why this concept is useful, refer to Figure 2.7. The natural grasp plane orientation we found with these three faces is such that it will be very difficult to grasp the object with the three fingers contacting their respective faces at the intersection of the object and the plane. The two upper fingers would be required to grasp at the edges of their faces.

This is a problem, because the example is not so different from that in

Figure 2.6, in which we have an extra degree-of-freedom in choosing our grasp plane orientation. For the case in Figure 2.7 we can characterize this similarity by noting that the grasp plane orientation actually does have a leeway of 90 degrees about the normals of the contact faces (compare the two upper plots in Figure 2.10). The plots show that we have an extra degree-of-freedom in choosing a grasp plane orientation here as well, although the effective coefficients of friction will vary a little as the grasp plane is rotated.

In Figures 2.8 and 2.9, where no qualitatively different grasp plane orientations will work, a leeway calculation will tell us that there is not much room to deviate from the natural grasp plane orientation. This is shown in the two lower plots of Figure 2.10.

Thus, from natural orientation and leeway information, we can come up with a set of *candidate* grasp plane orientations for each configuration of contact faces. In cases where all the faces are nearly parallel, such as in Figures 2.6 and 2.7, we will have a range of possibilities. In practice, we divide these into four to eight representative orientations. In other cases, such as in Figures 2.8 and 2.9, we have only one real orientation, with a small amount of leeway.

When we generate a feasible grasp with a grasp plane in one of these candidate orientations, we can use the leeway in the orientation to adjust the grasp plane and change the grasp. This may be necessary if we need to move the hand out of the way of obstacles around the target object, or out of the way of other parts of the object itself.

2.1.2 Offset

We now have a grasp plane orientation, and we need to compute its offset in space. For any given grasp plane orientation, there are two different approach directions possible, roughly in the positive and negative directions of the grasp plane normal. We find an offset (and hence completely define the grasp plane) for each of these approach directions.

Our primary concern in selecting a grasp plane offset is that the plane intersect each of the faces. If the grasp plane intersects a face too near a face edge, however, the valid contact regions will be cut off at that face edge. We want an offset that places the plane a nominal distance into each of the contact faces if possible. For this, we can simply find the overlapping ranges of offsets for the three faces and choose an offset at some nominal distance from the approach end of the overlapping range (see Figure 2.11). The distance we choose should be large enough that we do not have to worry about the fingers missing the face in this direction due to position error, and it should be small enough that the fingertips can reach the plane easily (barring other protrusions of the object into

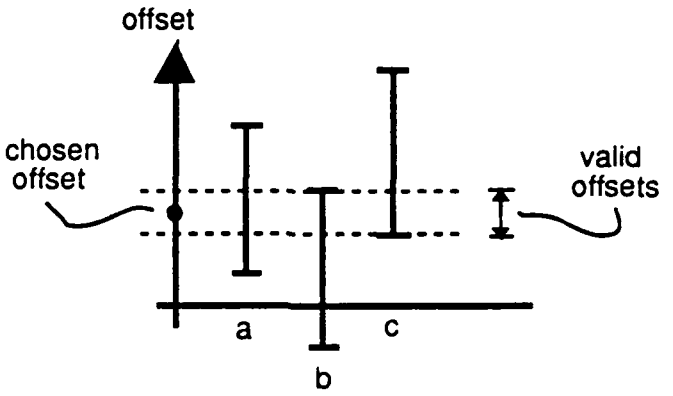


Figure 2.11: Choosing an offset for our grasp plane and faces a, b, and c.

the interior of the hand; we check this in the *Feasibility* chapter).

2.2 Focus Point

For a given grasp plane, we need to find a set of fingertip contact points. We can achieve this by first choosing a focus point for the grasp plane. This is the point at which the lines of force at the three contacts will intersect. For any focus point, we can calculate valid *contact segments* on each of the contact edges in our grasp plane from which the contact forces can point toward or away from the focus point without the object slipping. This is done by projecting the friction cones of each of the edges (using the effective coefficient of friction at that edge) from the focus point back onto the edge (see Figure 2.12). Although we will be refering to the contact *segment* of an edge, this can actually be composed of several disjoint co-linear segments if it is formed by taking a cross-section of a face that is not a convex polygon. The size of a contact segment is defined as the size of the largest of its segments. We define a *potential contact segment* as the projection of the friction cone of a contact edge from the focus point back onto the line formed from the edge. The contact segment is the intersection of the potential contact segment and the contact edge. Our goal here is to maximize the size of the smallest contact segment.

The method for finding the focus point is slightly different for different configurations of edges:

Nearly parallel: If the edges are parallel or nearly parallel, a focus point at infinity is chosen. The lines of force for the edges will be parallel and point along

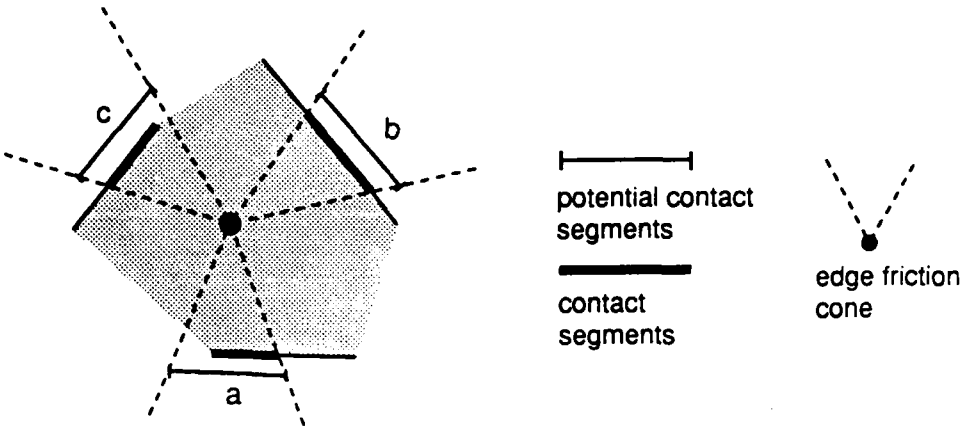


Figure 2.12: We can find the segments for each contact edge from which our force can point toward a given focus point without slipping by projecting the edge friction-cones from the focus point back onto the edges. The sizes of the friction-cones are calculated using the effective coefficient of friction of the edge.

the line that is the best fit to the three edge normals. This line can be obtained by negating the normal that is opposing the other two and then averaging the three resulting normal vectors (see Figure 2.13). The valid contact segments span the entire lengths of the edges.

Convex: If the edges can form a convex polygon. A focus point inside this polygon is chosen. As stated above, the placement of the focus point should maximize the size of the smallest contact segment. Assume that the edges are all infinitely long. Then we can achieve this by placing the focus point equidistant from the three edges. Because the normals of the edges cannot lie in the same 180 degrees of orientation, moving the focus point in any direction decreases its distance to some edge, and thus decreases the size of the valid contact segment on that edge.

The problem is more difficult if we take the lengths of the edges into account. Consider Figure 2.14. This figure shows equations for the size of a valid contact segment on the edge shown as functions of focus point position in each of six different regions. The regions are bordered by lines with the slope of the effective friction cone on that edge. μ' is the effective coefficient of friction of the edge, l is the length of the edge, and $[xy]^T$ represents the focus point position in the coordinate frame illustrated.

To maximize the size of the smallest contact segment on the three edges,

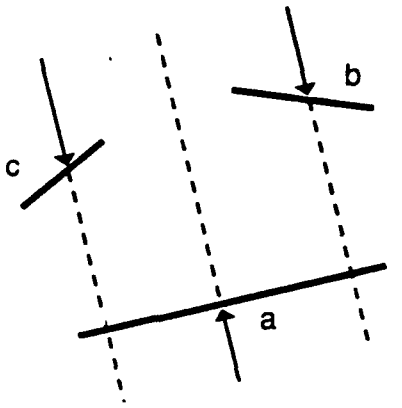
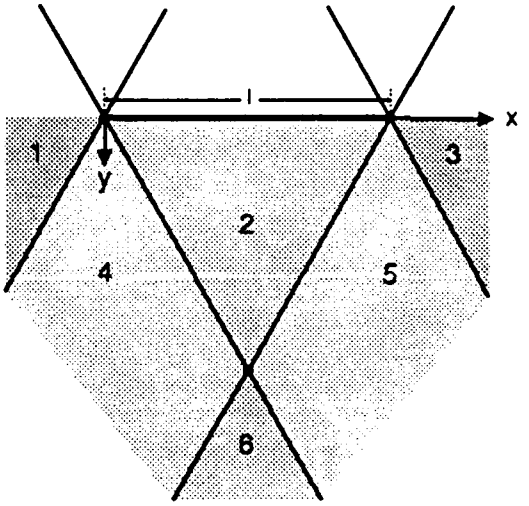


Figure 2.13: If the edges are nearly parallel, our focus point is at infinity, and the line of force is the best fit to the three edge normals.



		Size
1,3	=	0
2	=	$2\mu' y$
4	=	$x + \mu' y$
5	=	$(l - x) + \mu' y$
6	=	l

Figure 2.14: Characterizing the size of a contact segment as a function of focus point position for a real edge.

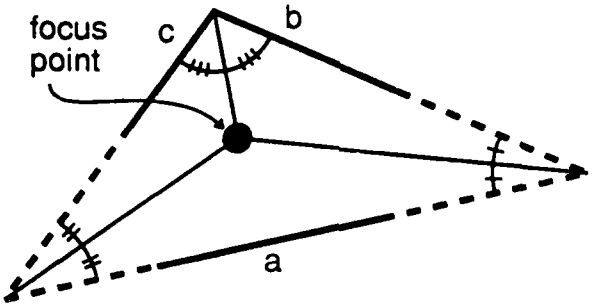


Figure 2.15: If our edges are convex, we place the focus point equidistant from the three edge lines.

we can optimize within each of the non-zero $4 \times 4 \times 4$ regions. In our examples, however, we use an approximation to this solution, choosing a focus point equidistant from all edges as if the edges were infinite. Figure 2.15 shows an example. As indicated above this method does not in general give us the largest contact segments on the real contact edges. There could even be an edge that has no valid contact segment.

Concave: If our contact edges must form part of a concave polygon, then a focus point outside these edges is chosen (Figure 2.16). Again, we would like to maximize the size of the smallest valid contact segment. In placing the focus point, only the concave edges are considered. The third, opposing edge will have a much larger potential contact segment, and we assume that there will be a sufficiently large intersection between this potential contact segment and the edge itself.

Again, to solve this problem correctly, we could divide the half planes formed by the two concave contact edges into six regions and search the resulting 4×4 space of non-zero regions to maximize the size of the smaller contact segment. We would also need to check that the size of the third contact segment is at least the size of the smaller of the two others, and if not, solve the $4 \times 4 \times 4$ problem as in the convex case above.

In our approximate solution here, we attempt to equalize the two contact segments by placing the focus point on the bisector of the two concave contact edges, and we attempt to maximize these segments by choosing our point on the bisector where one of the contact segments first hits the outer vertex of its contact edge (that is, when one of the contact segments stops expanding as we move up the bisector line and begins to shrink, Figure 2.16).

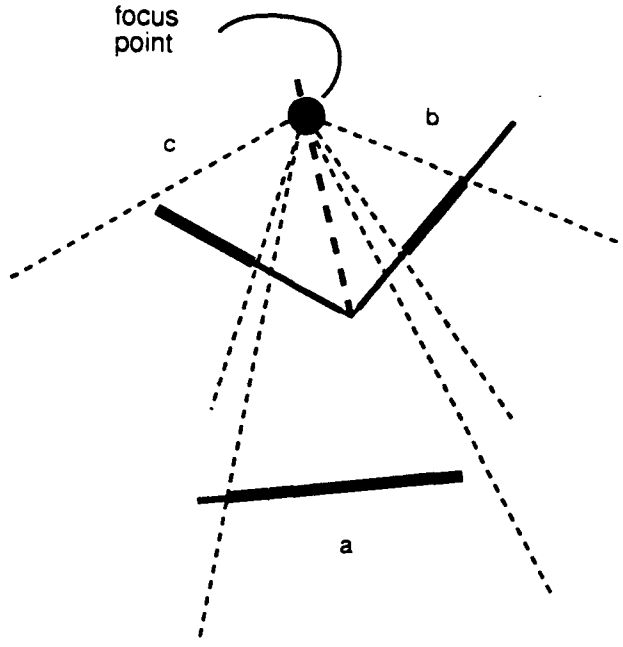


Figure 2.16: If we have concave edges, we place the focus point on the bisector of the two concave edges at the point where one of the contact segments first reaches the outer vertex of its contact edge (when moving up the bisector). The contact segment of the third, opposing edge should always be large.

This, of course, is not entirely correct, as the contact segments may not be equal (the potential contact segment for one or both contact edges may be larger than the contact segment), and the smaller contact segment will often not be maximal, since we are constraining the focus point to be on the edge bisector. This is a reasonable approximation, however, and it is easily computed.

2.2.1 Weighting the Distances

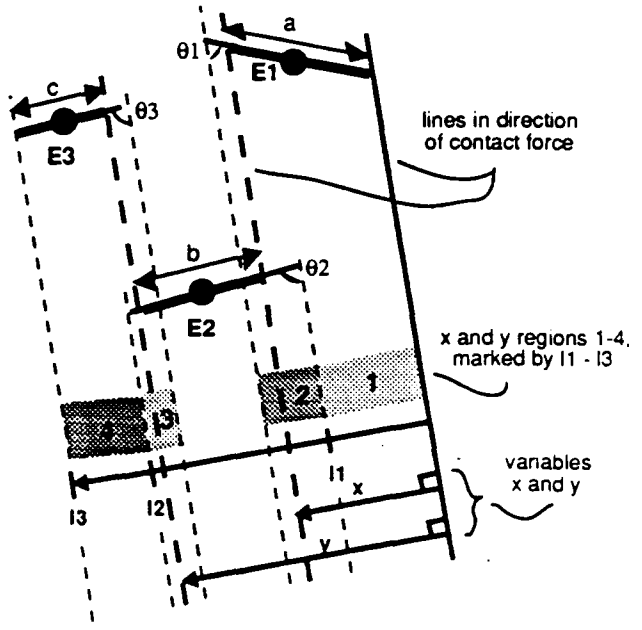
In calculating the focus point for the convex and concave sets of edges, we have tried to equalize distances from the point we are selecting to the contact edges involved. This has been in the interest of trying to produce equal valid contact segments on the edges. If the effective coefficients of friction on the contact edges are not equal, however, it does not work to simply equalize the distances to the contact edges. We need to weight each distance d_i by μ_i, the effective coefficient of friction of edge i, to produce d'_i. Then, if we equate the d'_i ($d'_1 = d'_2 = d'_3 = d'$), the potential contact segments will all have length $2\mu_i d_i = 2d'$.

2.3 Contact Points

At this point, the grasp plane and focus point have been found. The last step is to choose the contact points for the grasp.

Nearly parallel edges: If our contact edges are nearly parallel, we can in theory place the contact points anywhere on the contact edges. The focus point is at infinity, and the contact forces will be parallel, so we want to produce a counterbalancing grasp. That is, in the interest of balancing the torques on the object, we need to ensure that the upper fingers will be on either side of the lower finger.

Maximizing the size of the smallest valid contact segment is a two degree-of-freedom optimization problem. Consider Figure 2.17. E_1, E_2, and E_3 are the contact edges in the grasp plane. The placement of dividing lines parallel to the chosen direction of force determines the sizes of the valid contact segments, shown in the Figure as a, b, and c. The dividing lines themselves are shown at distances x and y from the origin shown in the Figure. The Figure also indicates the sizes of the valid contact segments as a function of the regions in which x and y are placed. These regions can be described as follows: Region 1 is the largest region containing edge E_1 and not containing edge E_3; Region 2, the largest region containing both edges E_1 and E_2; Region 3, the largest region containing both edges E_2 and E_3; and Region 4, the largest region containing edge E_3 and not containing edge E_1. Regions 2 and 3 can, of course, overlap.



$aSin(\theta_1) = x$	
$cSin(\theta_3) = (l_3 - y)$	
Region: x,y	$bSin(\theta_2)$
1,3	$(y - l_1)$
1,4	$(l_2 - l_1)$
2,3	$(y - x)$
2,4	$(l_2 - x)$

Figure 2.17: Finding contact points for nearly parallel edges.

Distances l_1, l_2, and l_3 indicate boundaries of these regions. The equations in the Figure can be used to find optimal contact segments for the edges and force direction selected.

Convex or concave edges: If our contact edges are convex or concave, we place each contact point at the center of the largest region of the valid contact segment formed by projecting the friction-cone of each edge back onto the edge (Figure 2.18).

Figures 2.19 through 2.22 show contact points calculated for the four representative objects introduced earlier. Note that the contact points shown are on the surfaces of the objects. Since the fingertips of the hand are not points but spheres (although the hand model used does not reflect this), contact points representing points of placement for the centers of these spheres should be found on the surfaces of the objects *grown* by the fingertip radius.

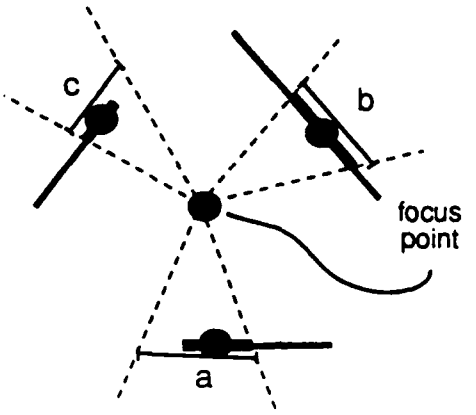


Figure 2.18: Finding contact points for a set of convex edges.

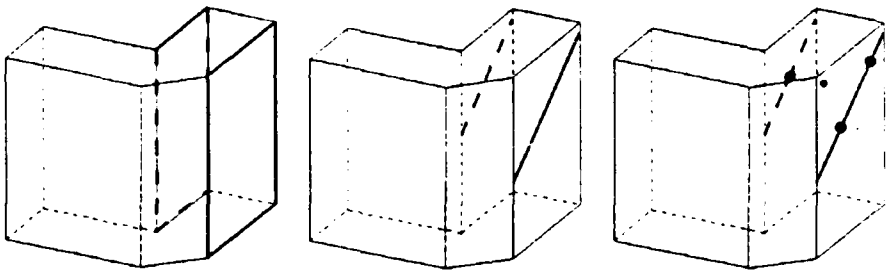


Figure 2.19: Faces, grasp plane, and contact points for a set of parallel faces.

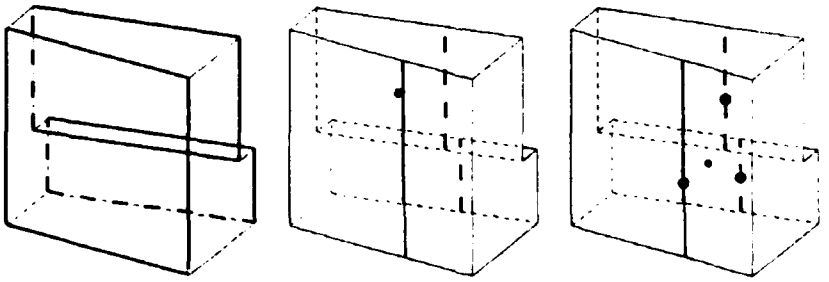


Figure 2.20: Faces, grasp plane, and contact points for a set of nearly parallel faces.

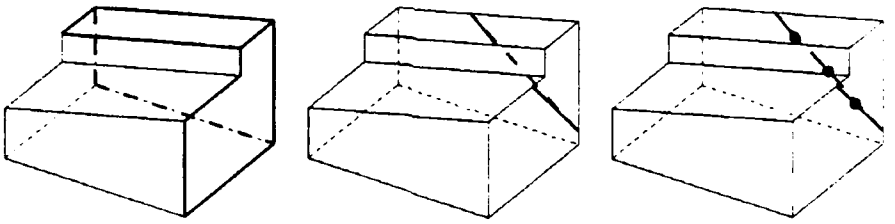


Figure 2.21: Faces, grasp plane, and contact points for a set of convex faces.

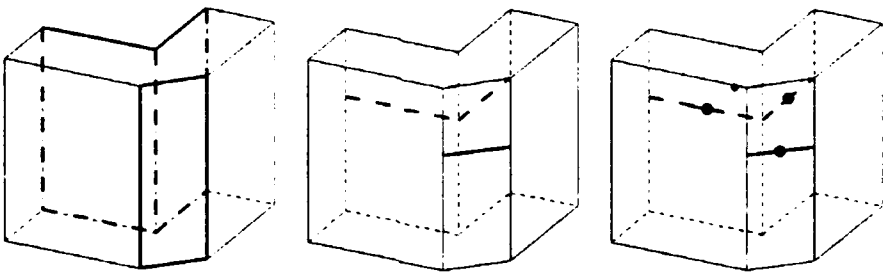


Figure 2.22: Faces, grasp plane, and contact points for a set of concave faces.

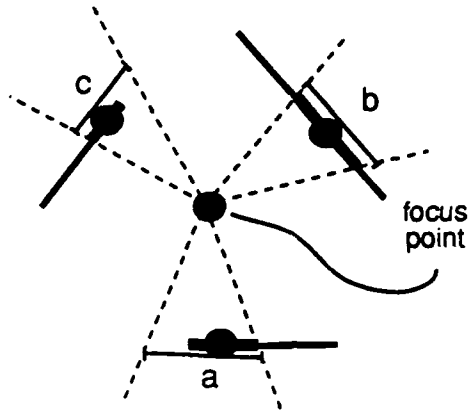


Figure 2.23: Contact regions for convex edges.

2.4 Contact Regions

A contact region is an area around a contact point where the stability of the grasp is preserved. It represents the error that can be tolerated in the placement of the fingertips. We will briefly look at one definition of contact regions *within* the grasp plane.

Nearly parallel edges: For a set of nearly parallel edges, we can describe our contact regions as any set of regions for which the upper finger forces are guaranteed to point to opposite sides of the lower finger force. Figure 2.17 shows an example allocation of contact regions in the grasp plane. As long as all the fingers are within their contact regions, we can apply parallel forces (that we can equalize), and we can balance the torques on the object (since the upper finger forces are on opposite sides of the lower finger force).

Convex or concave edges: If we have a set of convex or concave contact edges, we can form contact regions from the valid contact segments on the edges (formed by projecting the contact edge friction-cones from the focus point onto the edges). See Figure 2.23. As long as the contact points are within these regions, the fingertip forces can be directed through the focus point without the fingers slipping. This means that the torque on the object can be zero. And as long as the contact points are not all in the same half of a circle formed by the contact points (the contact regions can be clipped so that this does not occur), the forces can also be equalized.

These are both only definitions of contact regions within the grasp plane, however. A more interesting study of contact regions would be to:

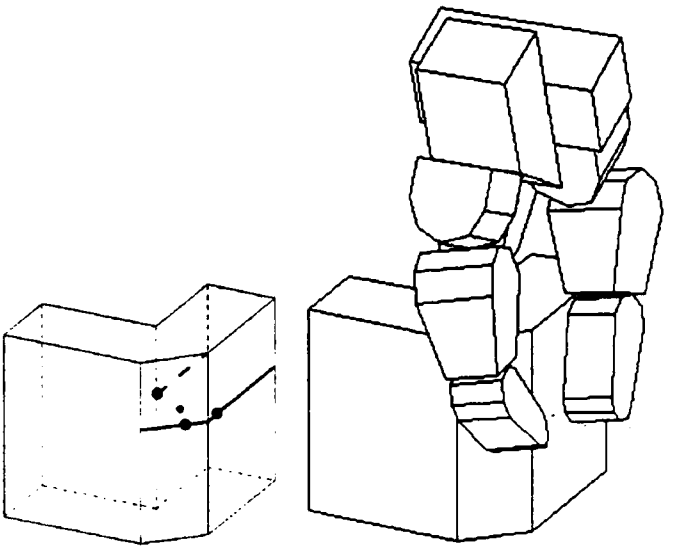


Figure 2.24: A grasp with the two upper fingers on parallel edges.

- Define the focus point (or focus points in the case of parallel forces) as fixed in the world.

- Define a control law for finger forces such as $f_i \propto \pm k_i \Delta x_i$, where Δx_i is the distance from the fingertip position of finger i to the focus point in world coordinates.

- Then, define contact regions to be independent regions such that contact in those regions, coupled with quasistatic execution of the control law, would result in a stable grasp with no slip at the contact points.

2.5 Examples

Figures 2.24 through 2.26 show three additional sets of contact points (and associated grasps) found using this algorithm. These are less conventional than the four representative grasps in that the two upper fingers of the hand do not directly oppose the thumb. Figure 2.26 even has the thumb and left finger on the same edge. All of these grasps are stable and kinematically feasible.

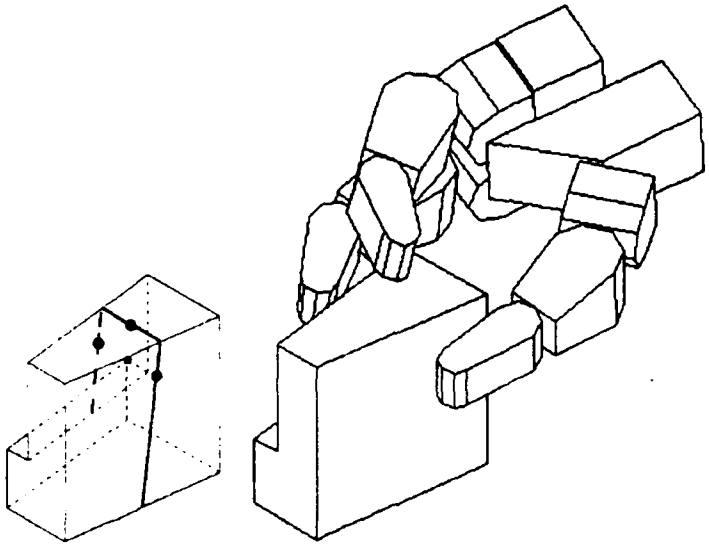


Figure 2.25: Another grasp with convex faces.

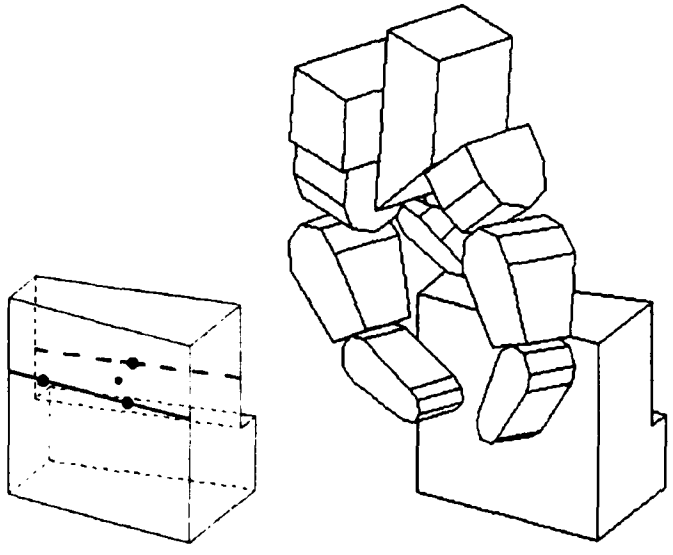


Figure 2.26: A grasp with the thumb and left finger on the same edge.

2.6 Summary

This chapter presented an algorithm for synthesizing a stable grasp of a given object. A finger-to-face mapping was given, and contact regions for the fingertips were produced. The goal of the algorithm was to maximize the size of the smallest contact region. The steps involved were as follows:

- Calculate a set of grasp planes, using the natural orientations indicated by the given face normals.
- Find a focus point for each grasp plane.
- Choose contact points for each focus point.

The algorithm is geared to a configuration with two fingers opposing a third, but, as we have seen in the example figures, there are many interesting finger configurations that fall within this simple characterization.

Chapter 3

Feasibility: The Wrist Configuration

In the previous chapter, we found sets of contact points on the target object that can be used to generate a stable grasp. In this chapter, we present an algorithm to generate a feasible grasp with the fingertips at these contact points. We may have to modify the given contact points to produce a feasible grasp.

For a grasp configuration to be feasible, the links of the robot must lie entirely within free space, and the joints of the robot must all be within their respective joint limits. We have already specified nine degrees-of-freedom of the robot in placing the contact points, so we have six degrees-of-freedom remaining. These can be specified by choosing a wrist position and orientation (a wrist configuration).

We generate a feasible grasp as follows:

- Before we attempt to find a good wrist configuration, we eliminate some contact configurations that are kinematically infeasible.
- Then we select a *starting configuration* for the wrist, based on a definition of an ideal grasp.
- If the starting configuration is not feasible (i.e. there are collisions between the robot and the world), we impose a quasistatic spring model on the joints of the robot and use a model of forces at the collision points to push the robot toward free space.
- If the wrist has moved from the starting configuration, we can adjust the grasp plane (within the leeway available before the grasp will slip) so that we will have a feasible hand configuration closer to the ideal hand

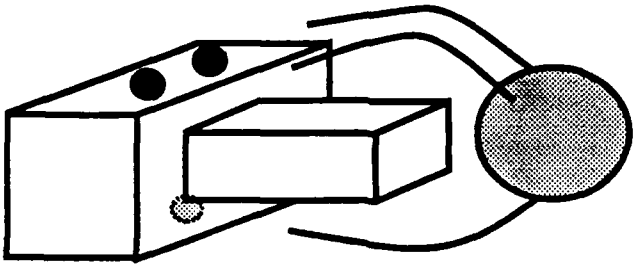


Figure 3.1: Contact points that are unreachable due to a collision of the object with the palm of the hand.

configuration. We find a set of contact points for the new grasp plane and repeat the process of finding a feasible wrist configuration for these contact points.

Each of these points is discussed in detail below.

3.1 Eliminating Infeasible Configurations

Since some of the steps in this chapter are computationally intensive, it is helpful to prune contact sets that are not promising. This section presents a method for eliminating some of the worst candidates. In particular, we want to eliminate grasps like the one illustrated in Figure 3.1. In this figure, the hand cannot enclose the object with its fingertips at the indicated contact points, due to a collision between the object and the palm of the hand. To check for this situation, we compute the amount that the object extends from the contact points perpendicular to the grasp plane from the finger approach direction. If this distance is greater than the distance from the palm of the hand to the fingertips, it is likely that the grasp will not be successful, and it is discarded.

Other simple kinematic checks could be devised, but this should eliminate some of the worst candidate sets of contact points.

3.2 Selecting an Initial Wrist Configuration

The initial configuration for the wrist, along with the set of contact points given for the fingertips, defines a grasp configuration for the robot. We would like this configuration to be as close as possible to an *ideal grasp*.

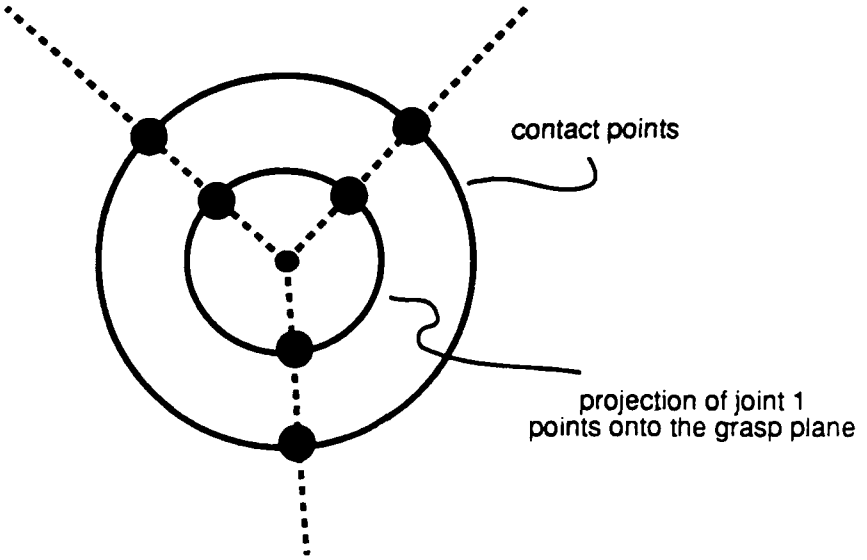


Figure 3.2: The first joints of the fingers projected onto the grasp plane.

Intuitively, an ideal grasp is one that keeps the joints of the hand far from their limits, and gives the hand a natural, streamlined profile. More formally, the following criteria are desired:

- The plane of the contact points (the grasp plane) is parallel to the plane formed by the first joints of the fingers (see Figure 1.1 for the location of finger joint one).
- The circle formed by the projection of the first joints of the fingers onto the grasp plane will be concentric with the circle formed by the contact points (Figure 3.2).
- A ray from the circle center through a contact point will intersect the corresponding projected first joint of the finger (Figure 3.2).
- The plane formed by the first joints of the fingers is as far from the grasp plane as the fingers can reach.

Figure 3.3 shows three ideal grasps for contact point circles of different radii. A grasp defined in this manner has equal distances from the first joint of each finger to the fingertip, and it has a streamlined hand profile. If the fingers have

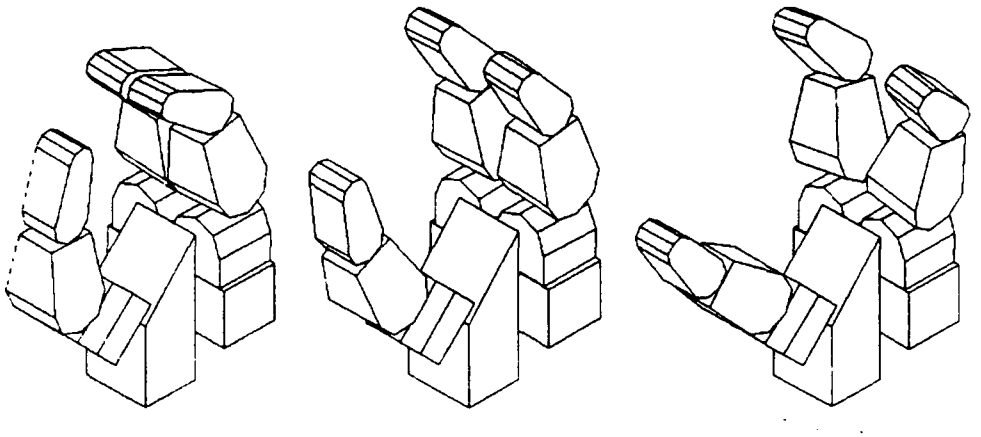


Figure 3.3: Ideal grasps for three different values of contact point radii.

to reach equal distances, we can more easily keep the finger joints within their limits. If the hand profile is streamlined, the volume swept out by the hand as it approaches the grasp will be minimized.

We want to choose a starting wrist configuration that will approximate the ideal grasp. Thus, we:

1. Place the plane of the first joints of the fingers parallel to the grasp plane. This constraint fixes two degrees-of-freedom in the orientation of the wrist.

2. Place the center of the circle formed by the projection of the first joints of the fingers onto the grasp plane concentric to the circle formed by the contact points. This fixes two degrees-of-freedom in the position of the wrist.

3. Orient the wrist so that the rays from the center of the circle through the contact points come *as close as possible* to intersecting the projected points. One definition of this is that the algebraic sum of the angles between rays through the contact points and the corresponding rays through the projected points should be zero. This fixes the remaining orientation degree-of-freedom.

4. Place the plane of the first joints of the fingers as far as possible from the grasp plane. This fixes the remaining position degree-of-freedom.

A grasp constructed in this manner deviates from the ideal grasp only in the distribution of the contact points about the center of the contact point circle. Figure 3.4 shows an example of a starting wrist configuration.

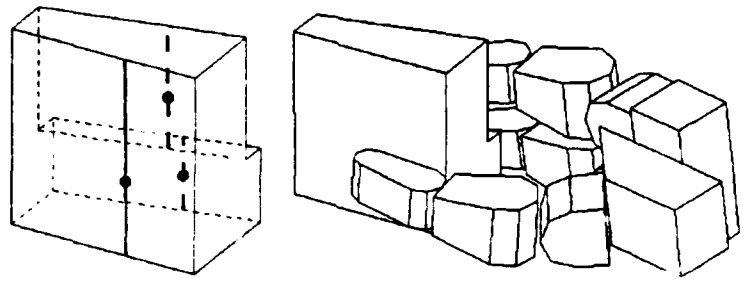


Figure 3.4: A starting wrist configuration for the contact points shown.

3.3 Eliminating Object-Robot Collisions

If the initial wrist configuration is not feasible, we use information on the location of object-robot collisions along with robot joint-limit information to attempt to push the robot into a feasible configuration. This is a local, potential field type method. It is used to eliminate collisions rather than to avoid them, since the robot begins in an infeasible configuration. In setting up a solution, we:

- Fix the fingertips in their current positions;
- Create a spring model of the joints with equilibrium at the current joint position;
- Collect points and directions of collision;
- Calculate the distances of the joints from their limits;
- Model the collisions as forces placed on the links at the collision points;
- Add a repulsive torque to any joint near its limit; and
- Calculate a new position for each of the joints in response to these forces and torques.

The new position becomes a new equilibrium point and we iterate until either there are no collisions or we give up.

In the paragraphs below, we derive the equation that governs the joint behavior. We start by noting the following relationship between instantaneous forces at the tip of the robot and torques at the robot joints:

$$\tau_{\mathbf{robot}} = \tau_{\mathbf{ext}} + J^T \mathbf{f}_{\mathbf{tips}} \tag{3.1}$$

where $\tau_{\mathbf{robot}}$ is the torque applied about the robot joints, $\tau_{\mathbf{ext}}$ is the torque applied about the robot joints by the environment and by its own joint limits, J^T is the transpose of the robot Jacobian, and $\mathbf{f_{tips}}$ is the force felt at the fingertips.

The repulsive force from collisions allows us to calculate $\tau_{\mathbf{ext}}$:

$$\tau_{\mathbf{ext}} = \sum_{l=links} J_l{}^T \mathbf{f_l} + \tau_{\mathbf{joint-limit}} \tag{3.2}$$

where $J_l{}^T$ is the Jacobian *to link l*, and $\mathbf{f_l}$ is the force on link l.

Joint limit torque is expressed as follows:

$$\tau_{j,joint-limit} = c_j(\frac{d_o}{d_j})^2 \tag{3.3}$$

$$|d_j| < d_o \tag{3.4}$$

where c_j determines the relative weight of joint limit external torques, d_j is the distance of the joint angle j from its joint limit, and d_o is the magnitude of the cutoff distance.

A spring response at each of the joints can be expressed as:

$$\tau_j = k\Delta\theta_j \tag{3.5}$$

where τ_j is the torque at joint j, $\Delta\theta_j$ is the deviation of the joint angle at joint j from equilibrium, and k is the stiffness of joint j. The equilibrium position for a joint is the current joint position. This helps eliminate useless internal motion.

Using the above equation, we get:

$$\tau_{\mathbf{robot}} = K\Delta\theta_{\mathbf{robot}} \tag{3.6}$$

$$\tau_{\mathbf{ext}} = K\Delta\theta_{\mathbf{ext}} \tag{3.7}$$

where K is the diagonal stiffness matrix incorporationg the stiffnesses of all the joints.

Combining the above equations gives us an expression for incremental joint motion in terms of the force felt at the fingertips:

$$\Delta\theta_{\mathbf{robot}} = \Delta\theta_{\mathbf{ext}} + K^{-1}J^T\mathbf{f_{tips}} \tag{3.8}$$

To find the force felt at the fingertips, we use the following equations relating joint velocities (or small joint motions) to cartesian velocities (or small cartesian motions):

$$\Delta\mathbf{x_{tips}} = J\Delta\theta_{\mathbf{robot}} \tag{3.9}$$

$$\Delta\mathbf{x_{ext}} = J\Delta\theta_{\mathbf{ext}} \tag{3.10}$$

This gives us:

$$\Delta\theta_{\mathbf{robot}} = \Delta\theta_{\mathbf{ext}} + K^{-1}J^T(JK^{-1}J^T)^{-1}(\Delta\mathbf{x}_{\mathbf{tips}} - \Delta\mathbf{x}_{\mathbf{ext}}) \quad (3.11)$$

$\Delta\mathbf{x}_{\mathbf{tips}}$ is used to compensate for any error in fingertip position that resulted from previous iterations (and our assumption of very small joint motion at each step). It is initially zero, since the fingertips are fixed. Rearranging, we get:

$$\Delta\theta_{\mathbf{robot}} = K^{-1}\tau_{\mathbf{ext}} + K^{-1}J^T(JK^{-1}J^T)^{-1}(\Delta\mathbf{x}_{\mathbf{tips}} - JK^{-1}\tau_{\mathbf{ext}})) \quad (3.12)$$

and $\Delta\theta_{robot}$, the incremental robot joint motion, is just what we want.

It remains to define how we derive forces on the links from collisions. In our world of polyhedral models, a collision is defined as either a point where an edge of an object intersects a face of the robot or a point where an edge of the robot intersects a face of the object. If we have a collision involving an *object* face, we use the face normal as the force direction at that collision point. If we have a collision involving a *robot* face, we use the negative of the face normal. The magnitude of the force is assumed constant for all collisions.

3.4 Adjusting the Grasp

If collisions or joint limits have forced the wrist away from the starting configuration, we can attempt to adjust the grasp plane to bring the final grasp closer to an ideal grasp. We would like to adjust the grasp plane orientation so that it is parallel to the plane formed by the first joints of the fingers in the feasible grasp configuration. The leeway of the grasp plane orientation limits the amount that the grasp plane can be adjusted, however. This step is only appropriate when the contact faces are nearly parallel and there is a large amount of leeway in the grasp plane orientation. We calculate a new grasp plane, new contact points, and a new feasible grasp with a grasp plane orientation that is closest to being parallel to the plane of the first joints of the fingers while remaining within the leeway of the grasp.

3.5 An Example

Figure 3.4 shows an example of a starting wrist configuration for a given set of contact points on an object. Figure 3.5 shows the world, with the robot feasibly grasping the object.

When the robot is grasping the object in the starting configuration, there are collisions between the robot wrist and the table, and between the upper fingers

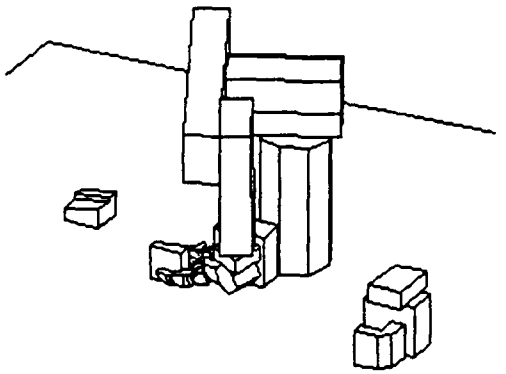


Figure 3.5: A feasible grasp.

and the block between the target object and the robot base. These collisions and equation 3.12 are used to push the robot into free space.

Figure 3.6 shows plots of all of the wrist parameters, by iteration, as the robot moves away from the block and the table. Wrist position (x, y, and z) is on the left, and wrist orientation (α, β, and γ) is on the right. α, β, and γ are the yaw, pitch, and roll angles of the wrist coordinate system. Starting with the world coordinate system and rotating γ about the world x-axis, then β about the world y-axis, then α about the world z-axis gives us the orientation of the wrist coordinate system.

In the world coordinate system, motion of the wrist in the z direction takes the robot up from the table, and motion of the wrist in the y direction moves the fingers away from the block they are hitting. The plots of wrist position in Figure 3.6 show that the robot first moves up from the table and then moves out away from the block. The position axes of the plots are drawn to the same scale to allow comparison of relative motions in the different directions.

The stiffness of the robot joints was set very high to generate extra points for the plots in Figure 3.6. This implies that the torques on the joints due to collisions were low, and that the robot only moved a small distance for each iteration of equation 3.12. In practice, we can use much lower stiffnesses. Figure 3.7 shows the number of iterations required for several different stiffness values. The number of collision points that exist between the model fingers and the block, and between the model wrist and the table before each iteration are shown. The error in fingertip position after each iteration is also shown. The fingertips are assumed to be fixed, but since we assume very small joint motions for equation 3.12, the fingertip positions will not be the same after a motion of

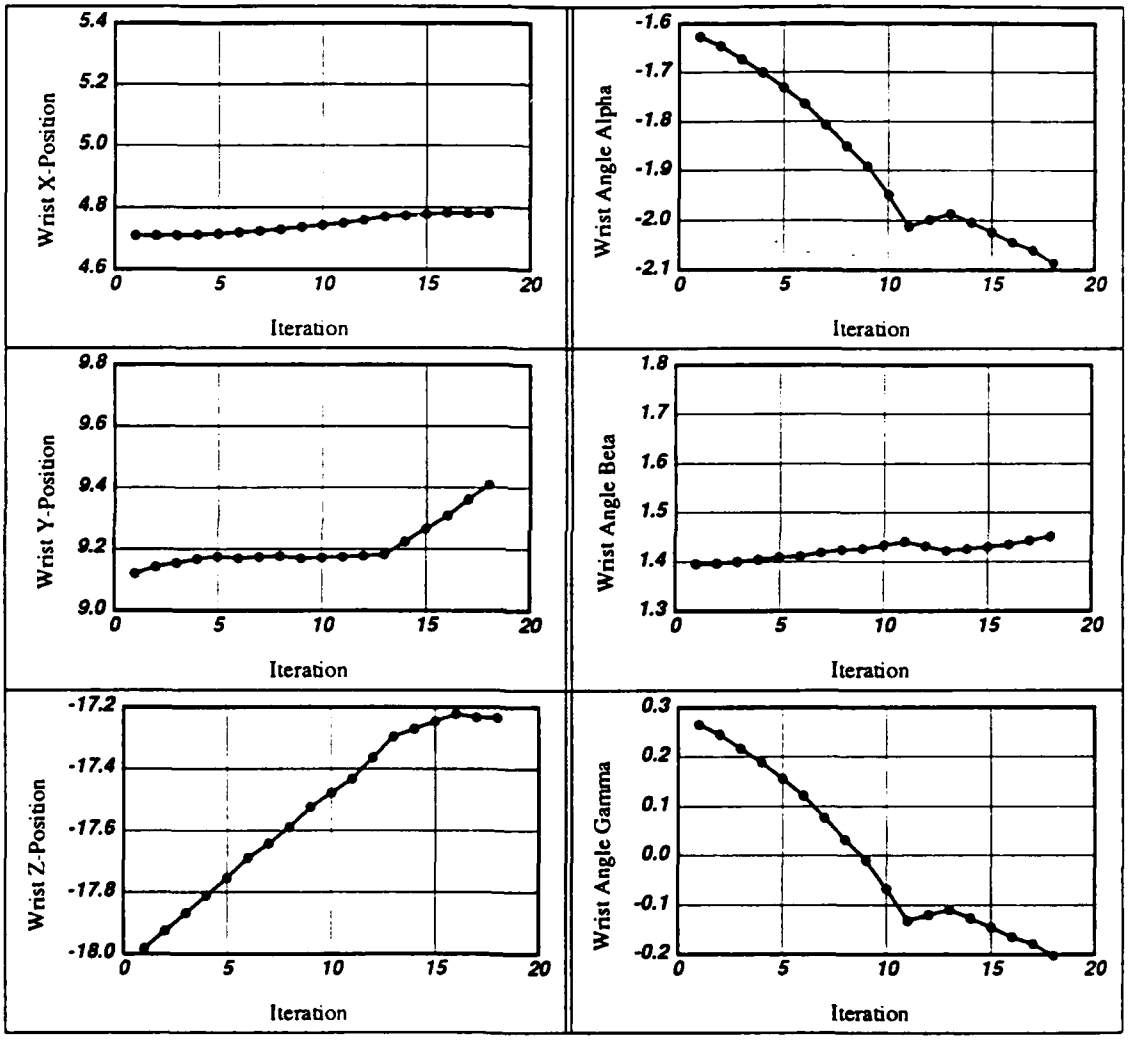


Figure 3.6: Plots of wrist position and orientation parameters by iterations of the collision avoidance step.

Stiffness factor	2.5			5			10			
Puma collisions:	12	0	0	12	3	0	12	3	3	0
Right collisions:	16	6	11	16	8	6	16	12	12	12
Left collisions:	6	0	0	6	0	0	6	0	0	6
Thumb collisions:	0	0	0	0	0	0	0	0	0	0
Fingertip error:	.17	.13	.20	.043	.047	.035	.011	.013	.014	.009

Stiffness factor	20						
Puma collisions:	12	7	3	3	3	0	0
Right collisions:	16	10	12	12	12	12	8
Left collisions:	6	6	0	1	6	6	0
Thumb collisions:	0	0	0	0	0	0	0
Fingertip error:	.0027	.0029	.0034	.0037	.0031	.0022	.0023

Figure 3.7: The table shows collisions, finger error, and the number of iterations required for different values of joint stiffness.

the robot as they were before the motion. The number given is the square root of the sum of squares of the individual fingertip errors in inches. Note that the fingertip error goes down as the square of the increase in joint stiffness (or the decrease in the size of the motions resulting from collisions at each iteration). For this example, at a stiffness factor of 2.5 only three iterations are required to generate a feasible grasp.

Figure 3.8 shows the new grasp plane found by adjusting the plane orientation to match the feasible hand configuration. This plane is within the orientation leeway of the faces. The figure also shows the starting wrist configuration with the contact points in the new grasp plane. This grasp is feasible. The world is shown in Figure 3.5.

3.6 Summary

In this chapter, we discussed an algorithm for generating a feasible grasp from a set of contact points. The algorithm is summarized below:

- Perform a quick kinematic feasibility check to eliminate grasps where the hand cannot enclose the object with the fingertips at the contact points.

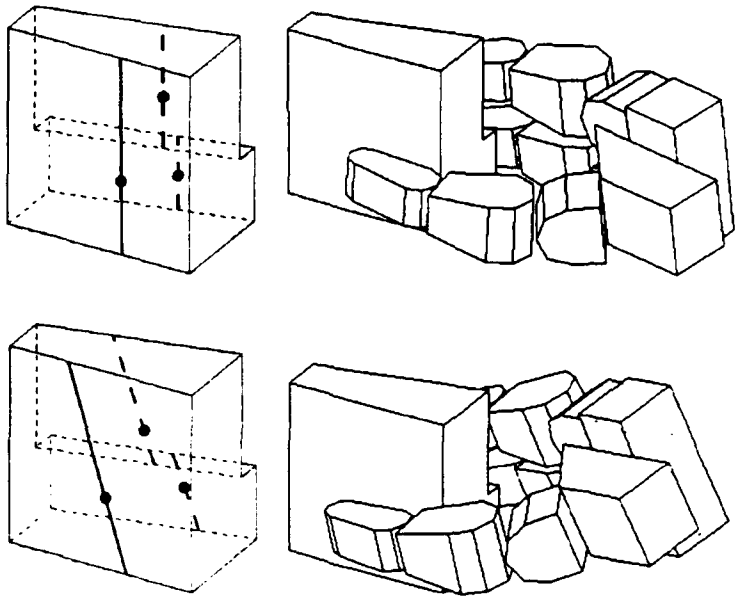


Figure 3.8: The starting wrist configuration for the contact points in the old grasp plane (top) and the new grasp plane (bottom).

- Find a starting wrist configuration based on a definition of an ideal grasp.
- Adjust that configuration away from collisions by modeling the robot joints as springs, the collision points as forces, and the joint limits as torques, and allowing the robot to move in response to the collision forces and joint limit torques.
- If there is sufficient leeway in the grasp plane orientation, adjust the grasp plane toward the feasible grasp, find a new set of contact points, and repeat the process.

The section *An Example* above showed the algorithm in operation. It works fairly well in general, but iterating to move the robot away from collisions can be slow. This example averaged about 10 seconds per iteration on a Symbolics 3650 Lisp Machine. We are in effect performing a directed search of a six-dimensional space at this step, however, so we can expect to pay in computation time to obtain a good search direction.

Chapter 4

Near Reachability: A Hand Path

In this chapter, we find an approach to a final grasp. A feasible final grasp configuration is given, and we want to find a clear path for the hand and arm from a point near the object into this final configuration. This step is important because the global path planner, which plans large-scale motions of the arm, is not accurate enough to guide the hand all the way into the final grasp configuration due to limitations in computing time and hardware. The grasp approach planner is an interface between the global path planner and the final grasp planner.

In planning a grasp approach, we assume that a small, straight-line motion of the hand will suffice. We initially plan only the hand motion. The corresponding motions of the arm will be small, and since the arm starts and ends in feasible configurations, we assume that intermediate collisions between the arm and the environment can be eliminated using local control.

The assumption of a straight-line approach allows us to approximate the volume swept out by the hand during the approach as a cylinder. We first find a free *approach direction* for that cylinder, and then we construct a *straight-line approach* for the hand in that direction, using local control (as in the *Feasibility* chapter) to modify the approach when there are collisions between the robot and the environment.

In the sections below, methods for finding an approach direction and for constructing an approach are discussed. Two examples are provided to illustrate the behavior of these methods.

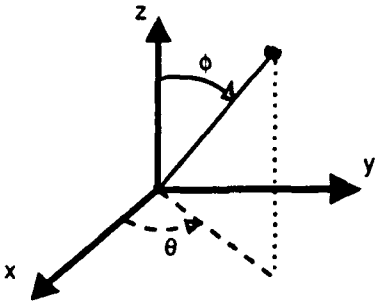


Figure 4.1: The coordinate system used for the grasp approach.

4.1 Finding an Approach Direction

This section is concerned with finding a clear approach direction for the hand. We assume that a small straight-line motion of the hand will suffice for the grasp approach, and we define a cylinder to represent the volume swept out by the hand as it makes this approach. A clear approach direction is defined as a direction in which this hand cylinder lies entirely in free space.

To set up the problem, we:

- Define a wrist coordinate system.
- Compute the swept volume cylinder for the hand.
- Grow the objects in the world by the cylinder radius and shrink the cylinder to a line.
- Define the clear approach distance for the hand as a function of cylinder orientation.

Then we search the space of cylinder orientations for a good approach direction.

4.1.1 The Method

The *wrist coordinate system* is based on the orientation of the hand in the final grasp configuration. It is pictured in Figure 4.1. We call the center of the circle formed by the first joints of the fingers of the hand the *wrist point.* This is the origin of the coordinate system. The z-axis is placed on the line between the wrist point and the center of the circle formed by the contact points (the

fingertips). It represents the current orientation of the hand. The placement of the x and y-axes is arbitrary. The wrist point will be the reference point for motions of the hand in the grasp approach.

We calculate a cylinder to approximate the volume swept out by the hand as it approaches the final grasp. One easy way to do this is to use the z-axis of the coordinate system defined above as the cylinder axis, and then to use the distance to the axis of the outermost point on the hand as the cylinder radius. A hand cylinder with this radius can fully enclose the hand in the final grasp orientation.

We base the hand cylinder at the origin of the wrist coordinate system. Since this is our reference point on the hand for motion during the approach, it will move along the approach cylinder axis. We can terminate the cylinder axis at the location of the point in the final grasp configuration. The cylinder will have a length representing the distance of the final approach. In practice we have used a minimum length of 5.5 inches. Finding a clear approach direction for the hand is approximated as the problem of finding an orientation in which this hand cylinder lies entirely in free space.

Now we grow all of the objects in the world by the cylinder radius, which allows the cylinder to be represented as a line. The problem has now been reduced to finding an orientation for which this line lies in free space in the world of grown objects.

Cylinder axis orientation can be represented by spherical coordinates θ and ϕ (see Figure 4.1). For every possible orientation of the cylinder axis there is an associated clear distance, which is the distance from the origin to the nearest intersection of the cylinder axis with a grown object.

Sometimes the origin itself will be inside a grown object. If this is the case, there is no simple way to directly compute a correct clear approach distance for the hand. To solve this problem, we move the origin into free space by placing it at the nearest point on the object surface, and we make this our new reference point for motion of the hand.

Any orientation in which the clear distance is greater than the minimum approach length is a good direction within this framework. There is a natural approach direction for any feasible grasp, however: the direction of the axis of the hand in that final grasp configuration. This is a good direction for an approach because the hand can enclose the object with its axis in this direction, and so it is likely that there will be a path for the fingers around the object. There can be problems reaching around the object if the approach is at too great an angle from this axis. The second example below illustrates this problem, which is an artifact of the straight-line approach requirement.

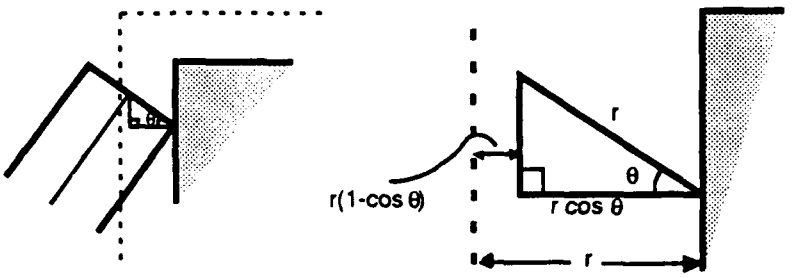


Figure 4.2: By growing all the faces outward by r in the direction of their face normals, we overcompensate.

We begin our search for a good approach line orientation with this natural orientation, at $\phi = 0$. If the clear approach distance is too small in this direction, we increment ϕ and check the clear approach distances for the entire range of θ at the new value of ϕ. If any of these directions have large enough clear approach distances, we take the one with the largest clear distance. Otherwise, we increment ϕ again and continue. The orientation choosen is at the smallest possible angle from the natural orientation (within the resolution used), and it is at the orientation that has the largest clear approach distance at this angle.

4.1.2 Innacuracies in the Method

The inaccuracies in the assumptions we have made when finding a clear approach direction can be summarized as follows:

1. The cylinder radius is not accurate. It varies with the angle of approach. It is only guaranteed to be sufficiently large when the approach angle is along the axis of the hand in the final grasp.

2. We grow objects by moving all the faces outward along their normals by the cylinder radius (r). This results in an overcompensation by $r(1 - \cos\theta)$ if the cylinder actually meets the face at angle θ (Figure 4.2). This overcompensation is even worse if the cylinder hits a grown object *edge*, especially if the faces that meet at that edge form a very acute angle.

3. The cylinder is a very crude approximation to the volume swept out by the hand.

4. We ignore the arm.

5. We may move the cylinder origin if it is inside a grown object. This makes the hand cylinder calculation very inaccurate and generally generates an approach in which the hand slides along the surface of the object from which we removed the origin. The first approach example below is such a case.

Approximations 2 and 3 could cause no path to be found. If this happens, we could use a smaller cylinder radius and try again. Approximations 1, 4, and 5 could cause us to find a path that results in collisions. If this occurs, we can attempt to modify this path using local control to produce a clear approach. This is covered in the section below.

4.2 Constructing an Approach

This section covers the approach itself. We have a clear approach direction, and we want to execute a straight-line motion in this direction.

The initial hand configuration is identical to the goal hand configuration, but offset by the approach distance (here 5.5 inches) along the approach direction. To fully specify motion of the robot along the approach, we specify both the wrist point position and orientation along the approach path, and the positions of the fingertips along the approach path. This constrains the full fifteen degrees-of-freedom of the robot.

4.2.1 Computing Linear Motion

Fingertip motion: Default straight-line motion of the fingertips is defined as follows:

$$\Delta \mathbf{x_f} = \frac{1}{t}\mathbf{w_f} \tag{4.1}$$

where $\Delta \mathbf{x_f}$ is the change in position of fingertip $\mathbf{f}$, $\mathbf{w_f}$ is the vector from the current fingertip position to the goal fingertip position, and t determines the step size. t steps will bring all the fingertips to their goal contact points.

Wrist motion: The wrist motion depends upon the fingertip motion. We cannot blindly assign straight-line constant velocity motion to the wrist, because this does not guarantee that the fingers can reach their current positions from the wrist point. For any set of fingertip positions, the position of the wrist can be defined uniquely as the point along the vector from goal wrist point to current wrist point at which the hand is fully extended. The wrist point is placed as far out as the fingers can reach.

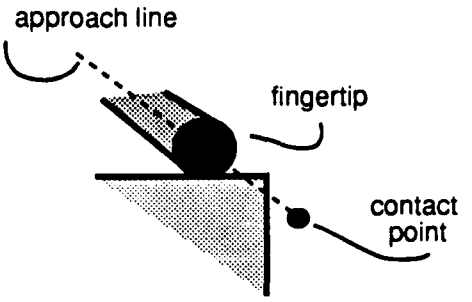


Figure 4.3: A straight-line approach for a fingertip may result in the finger colliding with some part of the target object before it reaches its final location.

To compute the orientation of the wrist, we use straight-line interpolation of wrist rotation about a fixed axis: w is the vector from the goal wrist point to the current wrist point.

- Define θ and **k** such that rotating the *world coordinate system* angle θ about fixed axis **k** would produce the current orientation of the *wrist coordinate system.* (See Craig [10], for example, for a derivation of θ and **k** from the rotation matrix taking one coordinate frame to another.)

- If the new wrist position is m**w** from the goal wrist position, the new wrist orientation is angle $m\theta$ about **k** from the goal wrist orientation.

Default motion of the wrist point is linear in that the wrist moves along a line from its current position to the goal, and the amount of rotation of the coordinate system about **k** is a linear function of the distance travelled along that line.

4.2.2 Avoiding Collisions

The above procedure does not take into account potential collisions of the robot with objects in the environment. In general this is a good assumption, because we assume that we have chosen a clear approach direction for the hand. There are two types of problems that may occur, however:

- Collisions of the fingertips with some part of the target object before they reach the contact point (Figure 4.3).

- Collisions of any other part of the robot with any object in the environment.

The first occurs when a direct line from a fingertip to its contact point intersects part of the target object. To avoid a collision, we allow the fingers to open wider, following an *object envelope* formed by growing the target object by some nominal distance. In our implementation, a finger can open in only one direction, the outward pointing normal to the edge it contacts in the grasp plane (refer to Figure 2.12 in the *Stability* chapter). This uniquely defines the projection of the default, straight-line fingertip path onto the object envelope. The fingertips will follow this projected path. This fails if the object is long in the direction in which the fingers can open, as they will not be able to open wide enough to clear the object.

The second type of collision occurs when some part of the robot other than the fingertips collides with some object in the world. This can happen due to inaccuracies in the swept volume cylinder approximations or due to ignoring the arm motion when finding an approach direction. In this situation, we keep the fingertips fixed, and use equation 3.12 as in the *Feasibility* chapter to push the robot back into free space.

4.3 Examples

In this section, we demonstrate the algorithm for finding an approach with two examples. In the first, the wrist point shifts to avoid a collision (Figure 4.4). In the second, the fingers open to avoid a collision (Figure 4.7).

Example 1: Figure 4.4 shows a frame-by-frame summary of the positions of the hand with respect to the target object and a second block during the approach. The arm is not shown for reasons of clarity. It is in free space throughout the approach.

In the second frame, the fingers are about to collide with the block. Local control equation 3.12 is used to adjust the robot's configuration to avoid the collision. Only one iteration of the control equation is required, as the fingers were going to just skim the object surface.

Figure 4.5 shows wrist position parameters x, y, and z, and wrist angle θ as functions of the distance remaining in the approach. The near-collision occurs with about 3.1 inches remaining in the approach. Note the large motion in the y-direction (away from the block) due to collision forces. θ in these plots is defined to be linear with respect to the distance remaining for the fingertips in the approach direction (and is plotted against this variable). A smoother

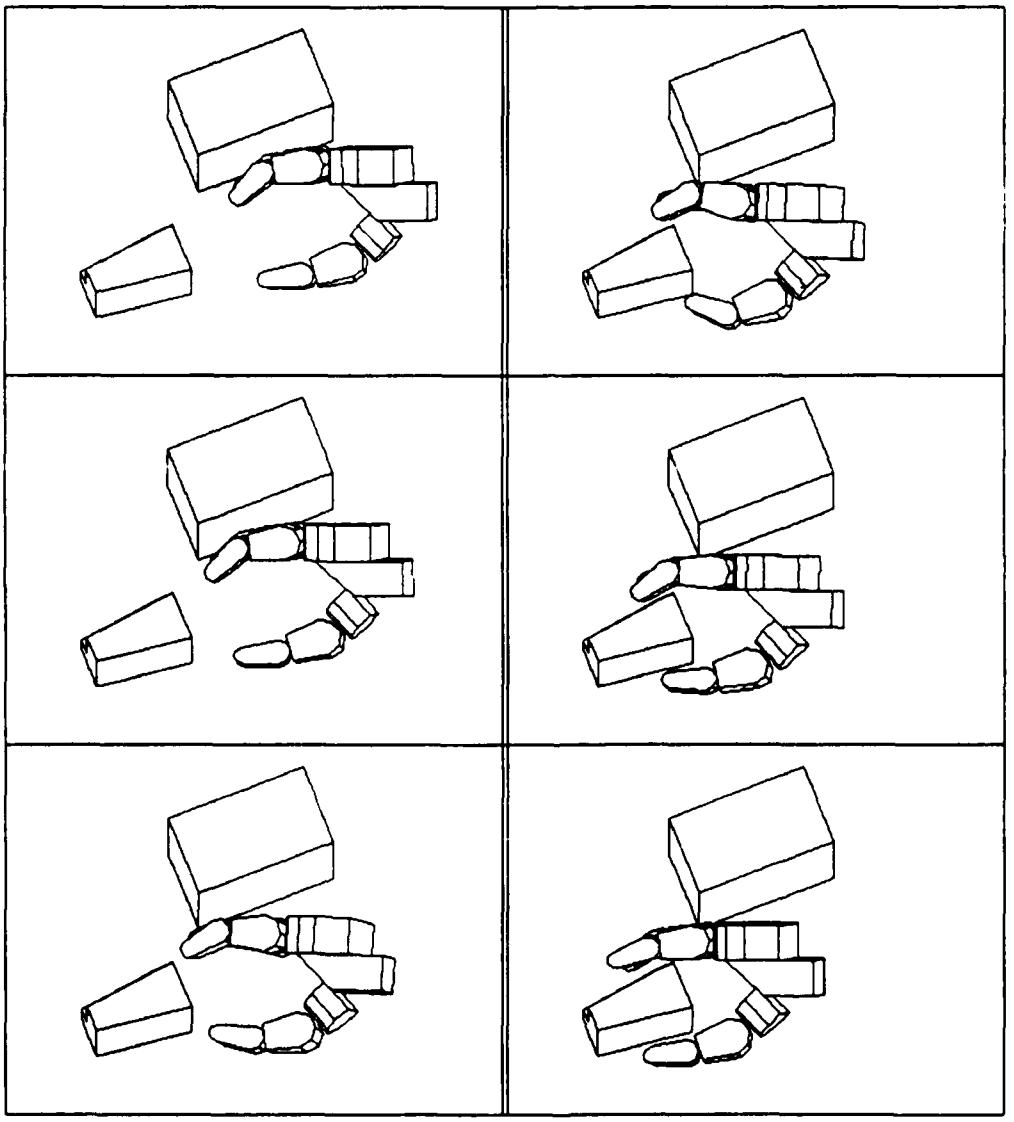


Figure 4.4: An approach sequence. The wrist moves off the straight line approach path to avoid a collision between the fingers and the block.

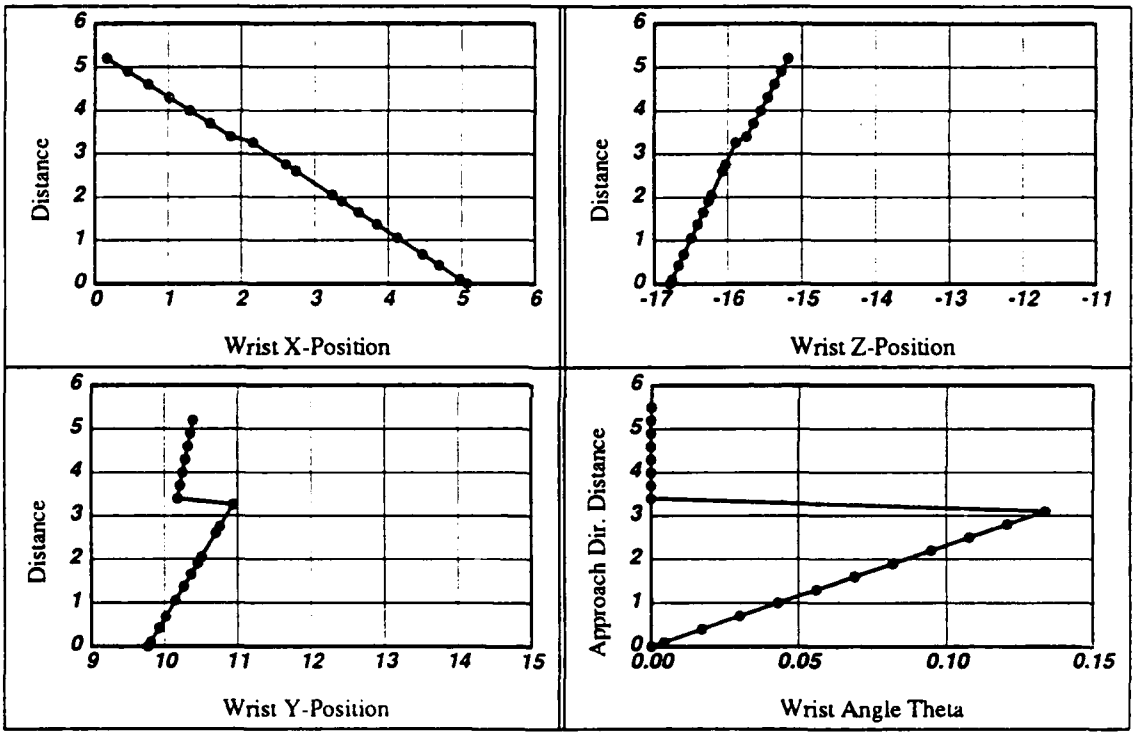


Figure 4.5: Plots of wrist parameters against distance remaining in the approach. **k** is $[-.867\ -.004\ .498]^T$, z is up from the table, y is away from the block, and the approach direction is roughly in the x-direction.

hand motion would result if it were defined as linear with respect to wrist point distance remaining, as suggested in the discussion above.

Plots of the x, y, and z coordinates against the distance remaining for the right fingertip are shown on the left side of Figure 4.6. The right finger opens a little in the y-direction to follow the envelope around the target object.

Example 2: The second example approach is shown in Figure 4.7. This is a straight-line approach with no collisions. It is included to show a more dramatic example of a finger opening to go around the target object envelope. This is best illustrated in the second frame of the bottom sequence of Figure 4.7. The thumb is forced out and around the stem of the "L". The wrist moves down to allow the fingers to open.

Plots of thumb position against distance remaining are shown on the right side of Figure 4.6. Note the large jump in thumb position in the x-direction. This is the direction along the stem of the "L". This approach would fail if the stem of the "L" were long, as the thumb would not be able to reach around it.

4.4 Summary

In defining the grasp approach, we first find a clear approach direction, and then construct a straight-line approach for the hand. If this approach causes collisions between any part of the robot and the environment, we modify it using local control equation 3.12, as we did when finding a feasible grasp.

We find a clear approach direction by approximating the volume swept out by the hand during the approach as a cylinder, and finding an orientation of that cylinder for which it lies entirely within free space. We grow the objects by the cylinder radius so that we can treat the cylinder as a line.

A straight-line approach is defined as follows:

- The fingertips move in a straight line toward their final contact positions, with constant velocity in the approach direction.
- The wrist point is placed on the line between its current location and goal location so that the hand is maximally extended.
- The wrist orientation θ about axis $\mathbf{k}$ is always a linear function of the distance from the current wrist point to the goal wrist point.

The fingers are capable of opening in one direction to go around the target object and avoid fingertip collisions. The wrist position and orientation are modified using local control equation 3.12 to avoid other types of collisions.

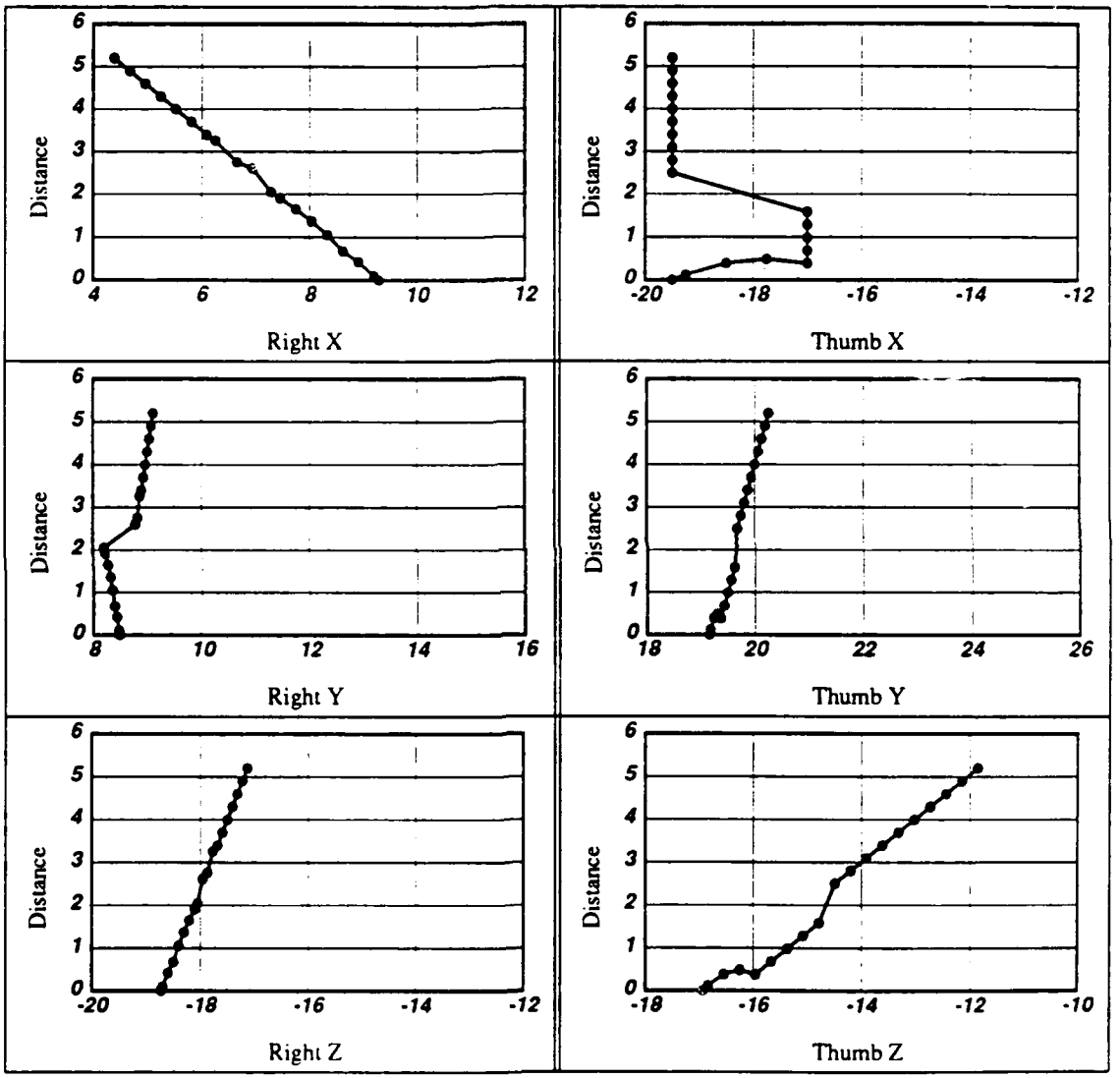


Figure 4.6: Right finger position against distance for the first approach, and thumb position against distance for the second approach.

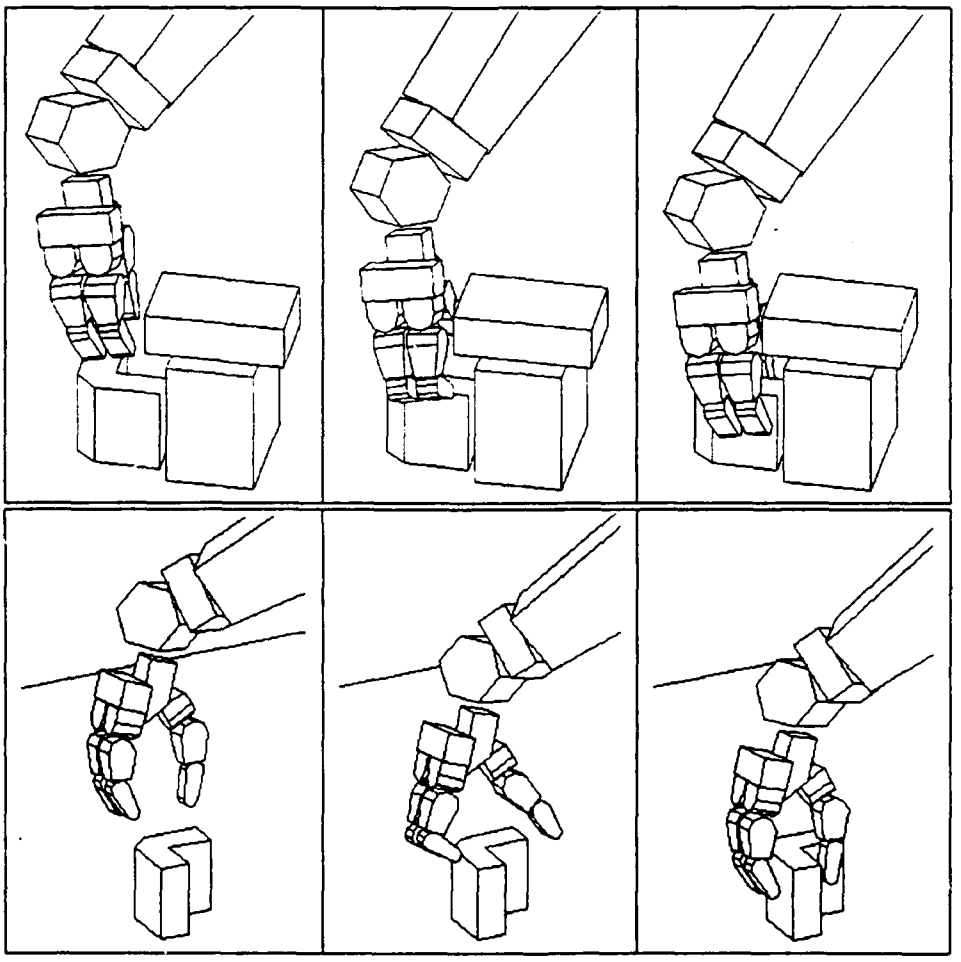


Figure 4.7: The second approach. The sequence on top is one view. The sequence on the bottom is another view, with the occluding objects removed for clarity. The thumb has to open a large amount to clear the object.

Chapter 5

Global Reachability: An Arm Path

At this stage, we have a grasp approach into a final grasp. This chapter describes how to find a path for the arm from an initial configuration to a configuration from which we can execute the grasp approach.

Since we are planning large-scale motions of the arm, we use a global path planner. To reduce the size of the problem, we will only search the space of motions of the first three joints of the robot. The positions of these joints determine the position of the robot wrist. The hand is considered to be a fixed payload. Since we have no control over the hand, we can only count on getting it *near* the target object with this planner. To move it to the final grasp configuration, we need the help of the grasp approach planner.

We describe a parallel algorithm to find an arm path. It consists of building a tessellated map of the configuration space of the first three joints of the robot, and then searching it for a path. Interesting parts of the algorithm are a precomputed map from configuration space to Cartesian space that we use to speed up the process of *building* the configuration space, and a method to speed up our search by in effect making the search space coarser while retaining low level connectivity information. The complexity of the algorithm in parallel is given by:

$$O[(\text{obstacles in the world}) + (\text{volume of arm}) + (\text{length of path start to goal})] \tag{5.1}$$

It is implemented on a Connection Machine [20], a SIMD (single instruction, multiple data) machine with 16K processors, each with 64K bits of local memory.

The flow of the algorithm is as follows:

- **Find free space in the world:** Tessellate the world (Cartesian space).

Each processor is a region. All regions check the world (or world model) to see whether or not they are occupied.

- **Find free configuration space:** Tessellate configuration space (the first three joints of the robot). Each processor is a region. All regions decide whether or not they are occupied by checking the Cartesian space covered by the arm in their range of configurations.
- **Search for a path through configuration free space:** Find connected regions from the starting configuration to the goal configuration.

In the paragraphs below, we discuss the algorithm in detail.

5.1 Finding Cartesian Free Space

For this part of the problem, we want to obtain a characterization of the regions in the real world (Cartesian space) that are free of obstacles. To do this, we first tesselate Cartesian space, assigning a region to each processor. We then loop through the objects in the world. A processor is marked as occupied if its region overlaps the volume occupied by the object under consideration. We can grow the obstacles so that we only have to check whether the midpoint of each region is inside the grown obstacle.

In parallel, this algorithm runs in time O(obstacles in the world). For environments with large numbers of objects distributed throughout the workspace, a speedup could be obtained by dividing space into large sections, each containing a set of objects, and then having each processor look only at the set of objects in its section.

Our world model could be replaced by a depth map, generated from a vision system (as in Brooks [4], for example) or a laser striping system mounted above the robot's workspace. In the absence of overhanging obstacles, such a system should give us enough information to find our path. The point-in-obstacle computation for checking whether a region is free is trivial: just use the region's xy coordinate to index into the array and then check whether its z-range lies above the level of objects measured at that point in the array.

5.2 Finding Configuration Free Space

In the previous section, we created a tessellated map of Cartesian free space. From this, we can compute a similar map of configuration free space. For every configuration of the robot, we must determine whether the robot is in free space.

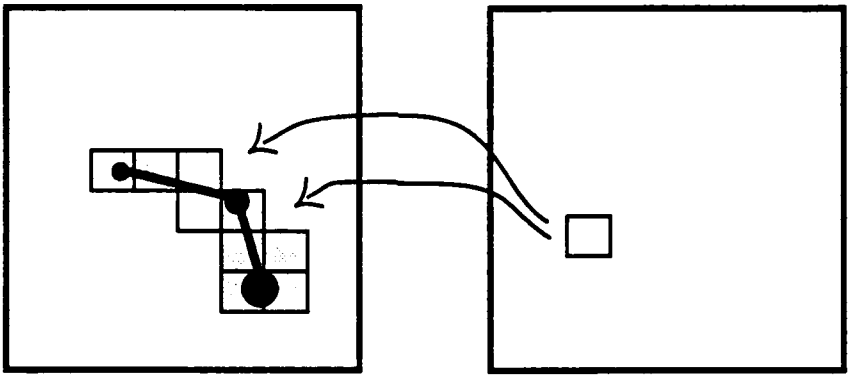


Figure 5.1: A configuration space element points to a list of Cartesian space elements that the arm covers in that configuration.

Two methods are presented for building the configuration space. In the first, each configuration checks corresponding *Cartesian space cells for obstacles.* In the second, occupied Cartesian space cells mark forbidden configurations. Both methods rely upon a mapping between the two spaces. The map is fixed for a given tessellation, so it can be computed offline.

Method 1: In the first method, we have available a precomputed map from configuration space regions to Cartesian space regions. This mapping gives for each configuration space processor the Cartesian space processors that the robot could overlap if it were in that range of configurations (Figure 5.1). Each processor goes through its list of Cartesian space processors, asking them if they are occupied. It stops and marks itself as occupied if it finds that its list includes an occupied region of Cartesian space. It concludes that it is free if and only if all of the cells in its coverage list are free. If all the cells in a processor's list are free, the arm is in free space over the range of configurations represented by the processor.

Method 2: If we were to propagate obstacle constraints in the other direction, *Cartesian space* processors would have a list of *configuration* processors that are blocked when there is an object in that Cartesian space region (Figure 5.2). All occupied Cartesian space regions would mark as prohibited all the configurations in their list. Free regions of configuration space would be those cells that remained unmarked.

Each method has advantages and disadvantages. With the first, excess work

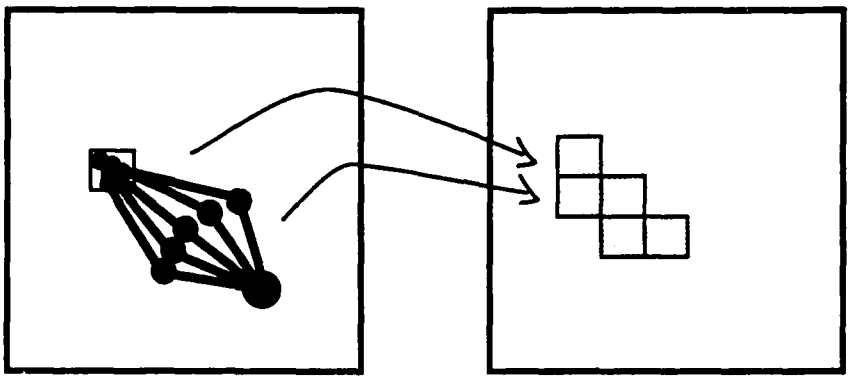


Figure 5.2: A Cartesian space element points to all configurations in which the arm intersects that region of Cartesian space.

is done when the configuration regions query unoccupied cells in their lists before reaching one that is occupied. In the second, work is wasted when Cartesian cells re-mark configurations that have already been marked prohibited. In the first, the bottleneck is multiple requests to the same area in Cartesian space. This occurs when a region of Cartesian space could block many configurations. If they all query the same cell at the same time, we have to wait for it to reply to all queries. In the second case, the bottleneck is the length of the lists. Again, a cell may block many configurations, and we have to wait for it to mark all of them.

Our implementation uses the first method, because the lists we have to store are a predictable, small size. If one wanted to work in the other direction, though, this problem could be minimized by adding a few special configurations. Instead of marking every element, a Cartesian space cell that blocks all configurations could mark *all*; or if it blocked the first link of the robot for some range of θ_1, the first joint angle of the robot, it could just mark that θ_1, instead of marking all $\theta_1\theta_2\theta_3$ configurations with that θ_1; or, similarly, if it blocked the second link for some θ_2 at some θ_1, it could mark that $\theta_1\theta_2$. Afterwards, this information could be propagated to the full configuration space. That way, the Cartesian space elements could just store the special elements in their list and not every configuration that those elements affect.

The map we use here is similar to the *shadows* of Shiller and Dubowsky [44], and to the links between the obstacle and arm configuration maps of Graf [18].

(Graf is trying to *learn* these links.) Shiller and Dubowsky present a different way of looking at the problem. In their version, there is also a Cartesian space map, but there are different copies of this map for each type of robot configuration. There are 4 of these for the first three links of the Puma, characterized as lefty or righty and elbow up or elbow down. Each Cartesian space location within one of these maps indicates a *wrist* position for the robot, which means that the maps are linked at robot singularities (when lefty is identical to righty, or elbow up is identical to elbow down). Since each wrist location, along with a configuration type, fully specifies the robot configuration for this three dimensional case, this is the equivalent of configuration space. Every cell in these maps can either maintain a list of elements that the arm covers when the wrist is at that point in that map (this corresponds to the configuration space lists above), or it can maintain a list of elements in its *shadow*, that is, a list of wrist locations that become illegal or blocked when it is occupied (this corresponds to the Cartesian space lists above). The advantage of this scheme is that we can better visualize the effects of the presence of objects, as we see in *Cartesian space* the points where the wrist (and hence the hand) cannot go if a particular cell is occupied.

Overall, the main limitation of the map procedure is due to the fact that the map must be precomputed (computing it at run time would be prohibitively time consuming: on the order of hours on the Connection Machine). There is only enough space to store a limited number of maps, and it takes much more space to store high resolution maps than to store low resolution maps. For example, at four inches of tip motion per division, we need $64K$ processors and at one inch of tip motion per division, we need $4M$ processors. This means that our possible resolutions of operation are very limited. Because of this, we cannot arbitrarily change our focus so that we can look more closely at an area in which it is difficult to navigate. Therefore, we can only rely on the configuration space map for coarse path information. In this implementation, we use a $64 \times 32 \times 32$ array, which gives us less than four inches of tip motion per division.

5.3 Searching the Configuration Space

Now that we have a characterization of free regions of configuration space, we perform a parallel search through connected configuration space from the initial arm configuration to the goal configuration. The most basic parallel search is to count distances of free space cells outward from the start until the goal is reached (Figure 5.3). Tracing the numbers back from the goal to the starting position gives us the shortest path between the points. The time to execute this

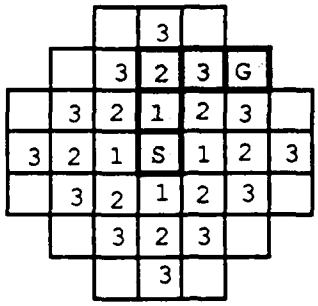


Figure 5.3: A simple parallel search in 2D.

algorithm (in parallel) is proportional to the length of the path, or $O(p)$.

In practice, we can increase the execution speed of the simple algorithm by a constant factor. In the two-dimensional example below, where the path is short, the difference is approximately a factor of 4. The basic idea is simply to group regions of free space together to form cells, and to find a path through connected cells. To achieve this, we first plant seeds at evenly spaced intervals among the configuration space regions. We then proceed to grow cells by counting outwards from all the seeds at once. If the counts from two different seeds overlap, a connection is formed between them. This is in some sense equivalent to increasing the size of our tessellation of the space, but here a cell is free if and only if its center region is free, and it is only connected to a neighbor if there is a path through the more finely tessellated space from its center to its neighbor's center. This scheme allows us to work at the coarser tessellation for the search while still finding a path if and only if one exists at the finer tessellation. This is guaranteed to work if we do not terminate the growing process before all connections are made, and if there is at least one seed in free space.

After the growing process terminates (either because of a threshold on the radii of the cells, or because every region in free space now belongs to some seed), the graph of connections between cell centers can be searched for a path using the simple algorithm described in the first paragraph above. The path is now the list of connected seeds.

If we threshold the radius of growth outward from the seeds by g (this enables us to ignore twisted, narrow paths in which large distances are required to connect two regions), if the spacing between the seeds is r, and if the time to find a path with the simple algorithm is p, the time to find a path with this

algorithm is:

$$\frac{p}{r} + g, \tag{5.2}$$

If the path is long, we can have a speed up of almost a factor of r.

Of course, the path is no longer complete to the resolution of the original tessellation, so there is a greater chance that the proposed path (a straight line course from seed to seed) will take the robot through some objects. We *can* recover a complete path from the information we have if we just keep track of distances and locations of collisions while growing the cells. Then, for any connection, we obtain a complete path between the two seeds by counting backwards towards them from the collision point we know. Although this does indeed give us a complete path at the level of resolution of a cell, it is at the expense of adding the path length between the seeds to the execution time.

A faster way to solve this problem is to add some local control to the motion of the arm between seeds to help us avoid the obstacles. This can be done very simply in configuration space by using a potential field type of algorithm. Each occupied cell produces a radial repulsive force with a decay factor due to distance. Since we are working in configuration space, this radial field represents a force proportional to the distance of the robot from an occupied configuration in joint space. Forces can be propagated outward indefinitely, if desired, to get a discreet representation of a potential field over the entire configuration space, but in practice it has been sufficient to do only two steps of the propagation. This scheme adds only two steps to the execution of the algorithm, and will work very well when the distances between seeds are at about the same scale as the curvature of the objects in the workspace, and when there are not concavities in the objects at that scale. If this procedure fails, we can always fall back on obtaining a complete path description as outlined in the paragraph above.

5.4 An Example

Figure 5.4 shows a two-dimensional global path planning example. We have a two link manipulator with two rotational degrees of freedom: one at the base, and one at the joint between the two links.

The top left pane of the figure shows this manipulator in the starting configuration (nearly vertical) and the goal configuration (lower and bent). The corresponding configuration space is displayed on the right. The lower point is the starting configuration, and the upper point is the goal configuration.

Free space is found by planting seeds at a seperation of 10 divisions, and growing each cell to a radius of 8 (center left pane). The seeds are displayed as

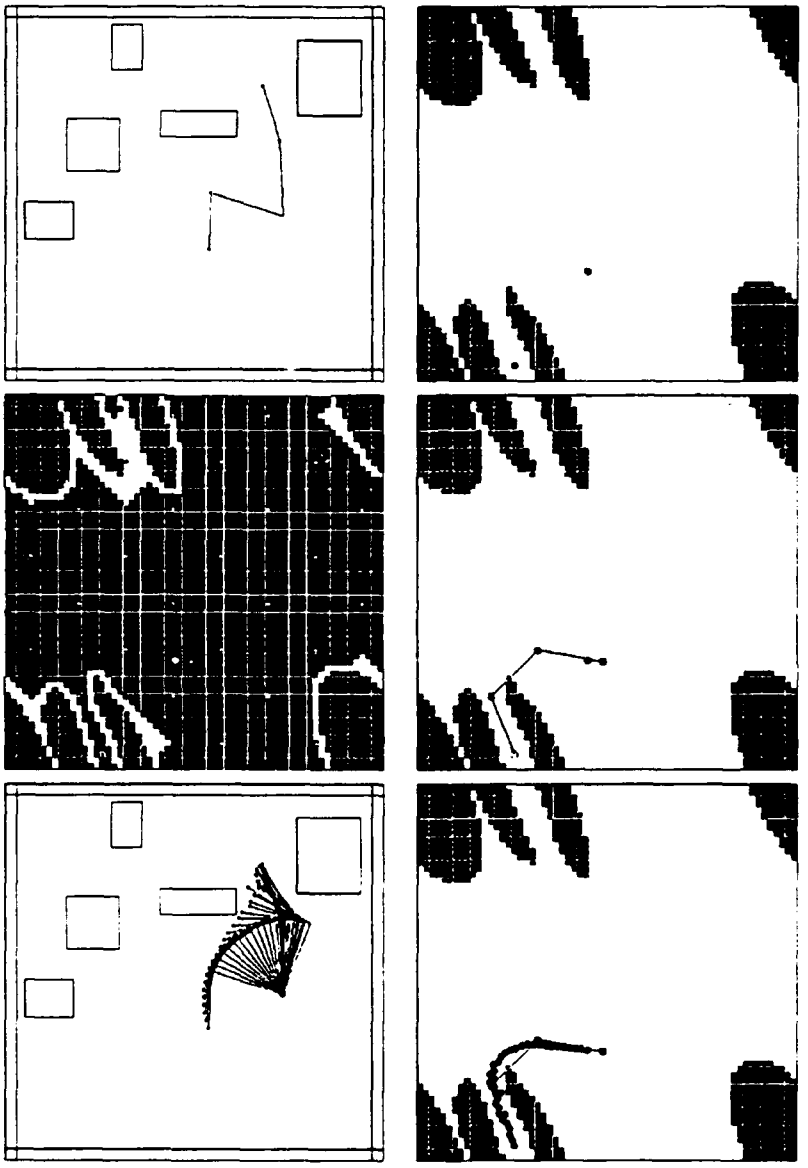


Figure 5.4: (TL) The manipulator in the initial (nearly vertical) and goal (lower, bent) configurations. (TR) Configuration space. The lower dot is the initial configuration; the upper one is the goal configuration. (CL) Free configuration space found by the growing algorithm. (CR) The path found for the arm. (LL) The motion sequence for the arm in Cartesian space. (LR) The configuration space motion sequence.

small white dots.

A path is found through the seeds, or cell centers (center right pane). Note that it passes through one of the objects.

The arm travels this path, using local control to avoid colliding with the corner of the block. The lower left pane shows a trace of the arm motion from the start to the goal configuration in Cartesian space. The lower right pane shows the same motion in configuration space.

Configuration space is built and searched, and a path is executed in approximately 5 seconds. The resolutions of the Cartesian and configuration space maps are 64×64.

5.5 Summary

In summary, we first find free space in the world by dividing it into equal regions, iterating through the objects in the world, and determining which of the regions are contained within the objects. We then find the configuration space of the first three joints of the robot by tessellating this space, and checking whether the Cartesian space regions that are covered by the arm in each configuration space region are free. We have a precomputed map from regions in configuration space to lists of regions in Cartesian space that aids us in this step. Once we have a representation of configuration space, we search through it. We can speed up the search by inserting seeds at evenly spaced intervals, determining if they are connected, and searching through the connections.

The main limitation in this step is in our parallel computing power. We are limited by our hardware as to how large a workspace we can cover at a given resolution. In particular, the size of the configuration space to Cartesian space map we can store, and the number of joints of the robot we can map are limited. We choose to map a three-dimensional configuration space at a resolution of less than four inches of wrist motion per division. For this we need a $64 \times 32 \times 32$ array of processores, or $64K$ processors. Since we have a fixed resolution, we only need to store a single map. There can be problems with this approximation if there are not wide paths available for the hand, but we assume that this is generally not the case.

Chapter 6

Summary and Discussion

6.1 Summary

The previous chapters presented methods for grasping objects with a Salisbury hand connected to a Puma arm. Grasps generated using these methods are stable, feasible, and reachable. The assumptions we make include:

- Using only fingertip grasps.
- Modeling fingertip contacts as hard finger contacts with friction.
- Using a world model, and modeling all objects in the world as polyhedra.

To compute the grasp, we first find contact points with their associated contact regions. Then we find a feasible grasp with the fingertips at those contact points. The default direction of the grasp is perpendicular to the plane of the contact points. If necessary, we adjust the contact points to help make the grasp feasible. Next, we generate a grasp approach. The default direction of the approach is the direction of hand axis in the final grasp. Finally, we find a path from the starting position of the robot to a point on the grasp approach line.

6.2 Extensions

In this section we discuss some possible extensions to the basic algorithm for grasping that overcome some of the system's current limitations.

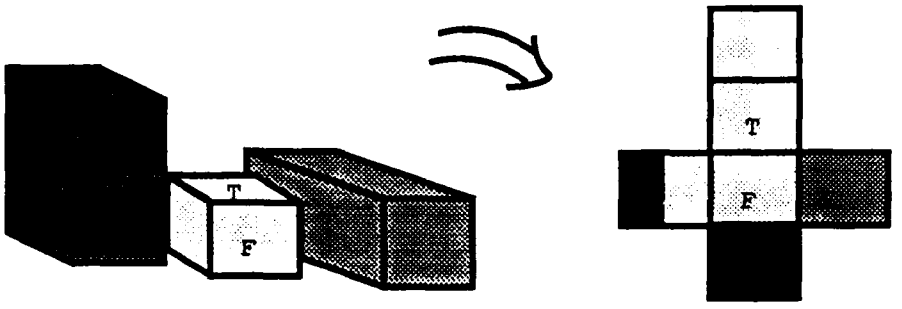


Figure 6.1: A two-dimensional example of potentially clear approach directions. We might want to choose to grasp the left and right faces.

6.2.1 Stable Grasp Extensions

Finger-to-face mappings: One step that was left out of the proceedure for finding a set of contact points on our given object was the selection of a finger-to-face mapping. We assumed that this information was provided. This is perhaps the least constrained part of the problem. To solve it correctly, however, we should take into account which configurations of faces can result in stable grasps, which of these stable grasps are feasible, and which of the feasible grasps are reachable; in effect, we would need to solve the entire grasping problem before we could even start. The worst we could do would be to simply enumerate all combinations of three fingers on three faces and sort out the good configurations at some later step.

Obviously, we cannot take all of the requirements for a good finger-to-face mapping fully into consideration when we pick our faces, but we can try to weed out some of the more obviously bad sets. (Recall that some weeding was also done in the feasibility step.)

Some possibilities include:

- To weed out some configurations of faces for which we might not even be able to find a set of contact points, we can eliminate from consideration all faces that have very little exposed area. Exposed area of a face can be determined by projecting all parts of objects very near the face onto the plane of the face, and then computing the area of the face that is free from projected obstacles (see Figure 6.1).

- To help ensure kinematic feasibility, we can look first for "preferred" configurations of faces. These are configurations that best fit the kinematics

of the hand. Here we would look for either two opposing, nearly parallel faces (as in Figure 2.1) or two nearly parallel faces opposing a third (as in Figure 2.2).

- To accomodate near reachability considerations, we can check for clear approach directions. We map out a space of clear approach directions by generating a cylinder that approximates straight-line motion of the hand, and finding the directions of motion for which this cylinder is free from collisions with objects in the world. Configurations of faces that have no reasonable approach direction can then be discarded. This step is computationally expensive, since we have to compute collisions between the hand cylinder and everything else over the two-dimensional space of cylinder orientations. It is also somewhat inaccurate, since at this stage we do not really know where to base the cylinder with respect to the target object. It may be a worthwhile check, however, if our object is complicated and we can eliminate many possible configurations with this step.

- To help ensure global reachability of a configuration of faces, we might want to take into account from which directions the Puma will be able to reach the object. This varies with respect to the position of the object in the workspace, but does not vary with respect to the environment (since we are not considering how obstacles affect reachability, only how the kinematics of the robot affect it). A map of reasonable directions could be computed once for a given robot, and referenced by object location to come up with a set of reasonable approach directions to consider (see Figure 6.2).

Arbitrary Objects: For objects that are not polyhedra, we could try to find some natural axis, perhaps by fitting a standard shape like a cylinder to parts of the object. Then we could attempt to place a grasp plane perpendicular or parallel to this axis (see Figure 6.3), and find contact points with appropriate normals in the cross-section of the object in this plane.

6.2.2 Feasible Grasp and Grasp Approach Extensions

One of the most computationally intensive parts of the entire system is the collision avoidance equation 3.12. It would be desirable to have alternative search procedures based on a simpler analysis of the local environment.

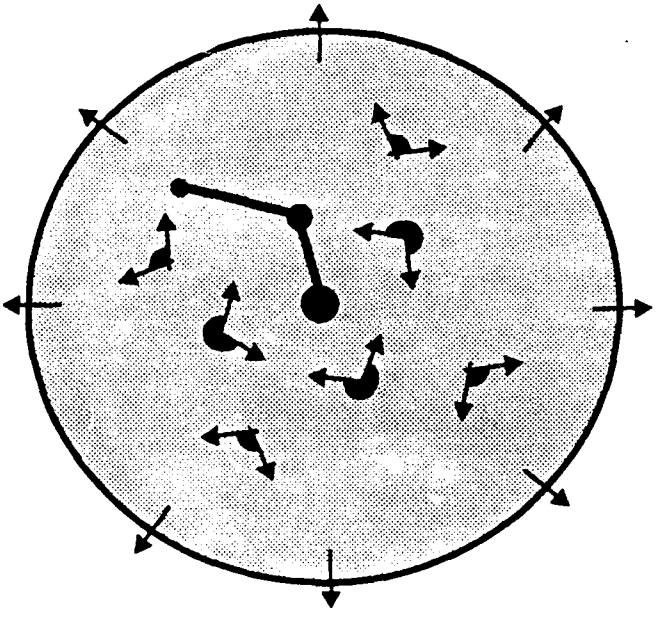


Figure 6.2: An example of a direction map for a two-degree-of-freedom planar manipulator.

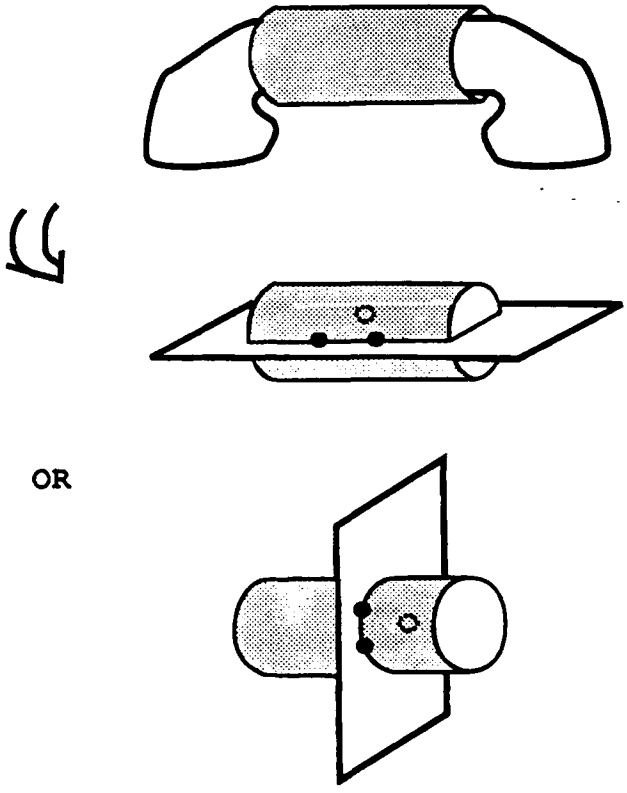


Figure 6.3: The handle of the telephone receiver can be modeled as a cylinder. Two grasp planes that we could generate for this cylinder are illustrated, one parallel to the cylinder axis, and one perpendicular to the axis. Only the first one would produce a feasible grasp in this case, because the rest of the receiver would be in the way of the second grasp.

6.2.3 Path Planning Extensions

If searching the space of the first three joint angles of our robot does not result in any possible motion to the goal, or if we have a more complicated robot, we may need to look for a path in a higher dimensional space. As we have seen above, there is clearly not enough memory in the current hardware to add even one more dimension without an unacceptably large increase in the time it takes to find a path through this space.

An approach that holds promise is to restrict the behavior of our robot, much as we did with the grasp approach earlier (see the *Near Reachability* chapter above). We could pick some limited set of types of global motions that we want to consider, motions that can be characterized with a few parameters. We could then search for a path through the space of these parameters.

If our arm is highly redundant, in that many configurations will produce the same wrist position, we may want to return to the method of first generating a path just for the hand or wrist through Cartesian space, and then using a potential field method to keep the rest of the arm away from obstacles. One way to do this is to model the robot as a simple physical system, reacting to forces applied to it by nearby objects. We can model the process as one of dragging the robot along by its endpoint, with the joints of the robot reacting as springs to potential forces produced on the links by nearby objects. This process would be similar to that used earlier to modify the wrist configuration in the final grasp to relieve stress from collision points, and to modify the straight-line approach to that grasp. Although any valid path for the robot will have a corresponding path through Cartesian space for the robot endpoint, we may have to try multiple wrist paths before finding one that works for the whole arm. We are not using any global information that relates to where the *arm* might be able to go.

If we do not want to parameterize the space of motions we will consider, we can include the influence of the hand and fingers in a different way. We can set up independent controllers of the arm, hand, and fingers. Depending on the task we are involved in, one of these would have priority, and the others would just try to keep their links free from collisions.

Some tasks are inherently arm driven. Consider our task of getting the arm to a point near an object. The main project in most cases will be to find a path for the arm, and the hand and finger configurations may be relatively unimportant. We can find the arm path as in this report. As we follow this path we can monitor the situation of the hand using proximity sensors (or a world model), changing the hand orientation when necessary and requesting a deviation in the path of the wrist if the hand is headed for an unavoidable collision or if it is moving too quickly. At an even finer level, we can seperately

monitor the fingers, adjusting their configurations as necessary to help them to avoid collisions and again requesting help from the hand if the job is too difficult. The usual problem of going down a wrong path due to using only local control can be partially avoided in this situation, because we know in advance the path planned for the wrist point of the robot.

A manipulation task, however, might be finger (or fingertip) driven. It might be defined by motions we want the fingers to make. In this type of situation we can use local control to guide necessary hand and arm motion. This control could be designed to keep all joints away from their limits and to keep all links from colliding with objects. We could also incorporate changes to our planned fingertip motions if there is a difficult situation (and if these motions are flexible).

Chapter 7

Conclusions

Grasping is a hard problem, where brute force methods would be too time consuming on any realizable computer system. Even if we specify the problem as three-fingered grasping for the Puma with the Salisbury hand, the search space is still large. The degrees-of-freedom that must be constrained are:

- Six for finding contact points.
- Six more for finding a feasible grasp.
- Fifteen to determine reachability.

And, of course, these subproblems are not really separate.

In order to cut down these problem spaces we made a large number of assumptions. These assumptions gave the arm and the hand a limited set of behaviors that we felt would be adequate for solving a wide range of grasping problems. The assumptions include:

- Using a deterministic algorithm for finding contact points. We attempt to produce maximal contact regions so that the intended grasp will have the greatest probability of succeeding, but a single set of contact points is produced. There is no flexibility in this sort of algorithm. We assume that there are enough good potential solutions that this method will work well in most situations.
- Using a deterministic algorithm for finding an initial wrist configuration. Here we attempt to take into account the kinematics of the hand. Since this often does not produce a feasible grasp due to collisions, we employ a directed search for a good configuration that is based on modeling the robot as a physical system with spring joints under stress due to the collisions. This is the only flexibility we have at this step.

- Using a straight line approach to the final grasp configuration. When this is not a good assumption, we use the same directed search as in the feasibility step to modify the approach.

- Using only three degrees of freedom of the arm joints to plan large scale motions of the robot through the environment. We have no backup plan for situations in which this assumption is not valid, although one possibility would be to incorporate some level of hand control, such as searching for paths for different ranges of hand orientation or using some sort of potential field control.

This set of constraints characterizes the space of grasps the robot can perform. The robot will successfully grasp the target object in situations where a grasp can be found in this space. Here we need an uncluttered world where we can maneuver near the object and make a straight line approach to a configuration of faces that somehow fits the hand geometry. A different set of constraints, or modifications of the constraints discussed in this report, could be used to handle a broader range of situations.

Bibliography

[1] B. S. Baker, S. J. Fortune, and E. H. Grosse. Stable prehansion with a multi-fingered hand. In *Proc. IEEE Intl. Conference on Robotics and Automation*, St. Louis, Missouri, March 1985.

[2] James Barber, Richard A. Volz, Rajiv Desai, Ronitt Rubinfeld, Brian Schipper, and Jan Wolter. Automatic two-fingered grip selection. In *Proc. IEEE Intl. Conference on Robotics and Automation*, San Francisco, April 1986.

[3] David Lawrence Brock. Enhancing the dexterity of a robot hand using controlled slip. Master's thesis, MIT Department of Mechanical Engineering, May 1987.

[4] Rodney A. Brooks. Planning collision free motions for pick and place operations. *The International Journal of Robotics Research*, 2(4), Winter 1983.

[5] Rodney A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13, 1983.

[6] R. C. Brost. Automatic grasp planning in the presence of uncertainty. In *Proc. IEEE Intl. Conference on Robotics and Automation*, San Francisco, April 1986.

[7] Stephen J. Buckley. Planning and teaching compliant motion strategies. Technical Report AI-TR-936, MIT Artificial Intelligence Laboratory, January 1987.

[8] John Canny. *The Complexity of Robot Motion Planning.* PhD thesis, MIT Artificial Intelligence Laboratory, 1987.

[9] John Canny and Bruce Donald. Simplified voronoi diagrams. Technical Report AIM-957, MIT Artificial Intelligence Laboratory, April 1987.

[10] John J. Craig. *Introduction to Robotics, Mechanics, and Control.* Addison-Wesley, Reading, MA, 1986.

[11] M. R. Cutkosky. Mechanical properties for the grasp of a robotic hand. Technical Report CMU-RI-TR-84-24, Carnegie Mellon Robotics Institute, 1984.

[12] Bruce R. Donald. Error detection and recovery for robot motion planning with uncertainty. Technical Report AI-TR-982, MIT Artificial Intelligence Laboratory, July 1987.

[13] Michael Andreas Erdmann. On motion planning with uncertainty. Technical Report AI-TR-810, MIT Artificial Intelligence Laboratory, August 1984.

[14] S. C. Jacobsen et al. The version 1 utah/mit dextrous hand. In H. Hanafusa and H. Inoue, editors, *Robotics Research: The Second International Symposium.* MIT Press, Cambridge, Massachusetts, 1985.

[15] B. Faverjon and P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Philadelphia, Pennsylvania, 1988.

[16] Bernard Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Atlanta, Georgia, 1984.

[17] R. S. Fearing. Implementing a force strategy for object re-orientation. In *Proc. IEEE Intl. Conference on Robotics and Automation*, San Francisco, April 1986.

[18] D. H. Graf. A neural controller for collision-free movement of robot manipulators. Master's thesis, Carlton University Department of Computer Science, September 1988.

[19] Hideo Hanafusa and Haruhiko Asada. Stable prehension by a robot hand with elastic fingers. In *Proc. 7th Int. Symp. Industrial Robots*, pages 361–368, Tokyo, October 1977.

[20] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[21] J. W. Jameson. *Analytic Techniques for Automated Grasps.* PhD thesis, Department of Mechanical Engineering, Stanford University, January 1985.

[22] J. Kerr and B. Roth. Analysis of multifingered hands. *The International Journal of Robotics Research*, 4(4), Winter 1986.

[23] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1), Spring 1986.

[24] Manja Kircanski and Miomir Vukobratovic. Contribution to control of redundant robotic manipulators in an environment with obstacles. *The International Journal of Robotics Research*, 5(4), Winter 1986.

[25] A. C. Klein. Use of redundancy in the design of robotic systems. In *2nd Intl. Symp. on Robotics Research*, Kyoto, Japan, 1984.

[26] C. Laugier. A program for automatic grasping of objects with a robot arm. In *Proc. 11th Int. Symp. Industrial Robots*, Japan Society of Biomechanisms and Japan Industrial Robot Association, 1981.

[27] Z. Li and S. Sastry. Optimal grasping by multifingered robot hands. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Raleigh, North Carolina, 1987.

[28] Tomás Lozano-Pérez. The design of a mechanical assembly system. Master's thesis, MIT Artificial Intelligence Laboratory, August 1976.

[29] Tomás Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(10), October 1981.

[30] Tomás Lozano-Pérez. A simple motion planning algorithm for general robot manipulators. Technical Report AIM-896, MIT Artificial Intelligence Laboratory, June 1986.

[31] Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, Patrick A. O'Donnell, W. Eric L. Grimson, Pierre Tournassoud, and Alain Lanusse. Handey: A task-level robot system. In *4th Intl. Symp. on Robotics Research*, Santa Cruz, CA, 1987.

[32] Tomás Lozano-Pérez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1), Spring 1984.

[33] Xanthippi Markenscoff and Christos H. Papadimitriou. Optimum grip of a polygon. *The International Journal of Robotics Research*, 8(2), April 1989.

[34] Matthew Thomas Mason. Manipulator grasping and pushing operations. Technical Report AI-TR-690, MIT Artificial Intelligence Laboratory, June 1982.

[35] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(4), 1987.

[36] Yoshihiko Nakamura, Hideo Hanafusa, and Tsuneo Yoshikawa. Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research*, 5(2), Summer 1987.

[37] Van-Duc Nguyen. The synthesis of stable force-closure grasps. Technical Report AI-TR-905, MIT Artificial Intelligence Laboratory, July 1986.

[38] C. O'Dunlaing, M. Sharir, and C. K. Yap. Retraction: A new approach to motion planning. In M. Sharir and J. Hopcroft, editors, *Planning, Geometry, and Complexity.* Ablex, Norwood, New Jersey, 1987.

[39] T. Okada. On a versatile finger system. In *Proc. 7th Int. Symp. Industrial Robots*, pages 345–352, Tokyo, October 1977.

[40] Jocelyne Pertin-Troccaz. On-line automatic programming: A case study in grasping. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Raleigh, North Carolina, 1987.

[41] J. K. Salisbury. *Kinematic and Force Analysis of Articulated Hands.* PhD thesis, Department of Mechanical Engineering, Stanford University, 1982.

[42] J. K. Salisbury and J. J. Craig. Articulated hands: Force control and kinematic issues. *The International Journal of Robotics Research*, 1(1), Spring 1982.

[43] J. T. Schwartz and M. Sharir. On the piano movers problem ii: General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4, 1983.

[44] Z. Shiller and S. Dubowsky. Global time optimal motions of robotic maniplators in the presence of obstacles. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Philadelphia, Pennsylvania, 1988.

[45] R. Tomovic, G. Bekey, and W. Karplus. A strategy for grasp synthesis with multifingered robot hands. In *Proc. IEEE Intl. Conference on Robotics and Automation*, Raleigh, North Carolina, 1987.

[46] M. Wingham. *Planning How tc Grasp Objects in a Cluttered Environment.* PhD thesis, University of Edinburgh, Scotland, 1977.